2021.09.25. 교육

나정휘

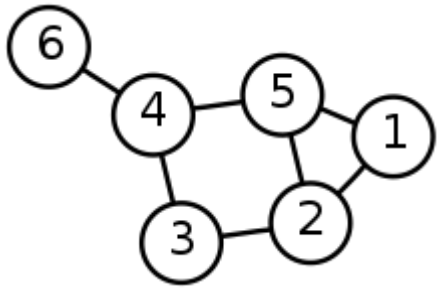
https://justiceHui.github.io/

목차

- Shortest Path
  - Dijkstra's Algorithm
  - Floyd-Warshall Algorithm
  - Bellman-Ford Algorithm
  - Shortest Path Faster Algorithm
- Minimum Spanning Tree
  - Prim's Algorithm
  - Kruskal's Algorithm
- 고인물 이야기 한 스푼...

Shortest Path Algorithm

용어 정의



- 보행(Walk): 정점과 간선이 교대로 나타나는 나열
  - 5 – 2 – 1 – 5 – 2 – 3은 보행임
- 트레일(Trail): 같은 간선이 여러 번 등장하지 않는 보행
  - 5 – 2 – 1 – 5 – 2 – 3은 간선 (5, 2)가 여러 번 등장하므로 트레일이 아님
  - 5 – 2 – 1 – 5 – 4는 트레일임
- 경로(Path): 같은 정점이 여러 번 등장하지 않는 트레일
  - 5 – 2 – 1 – 5 – 4는 4번 정점이 여러 번 등장하므로 경로가 아님
  - 2 – 1 – 5 – 3 – 6은 경로임
- 경로의 길이: 시작 지점에서 끝 지점으로 갈 때 거쳐 가는 간선 가중치의 합
- 최단 경로(Shortest Path): 길이가 가장 짧은 경로!

최단 경로 알고리즘

- 최단 경로를 찾는 알고리즘
- 문제 상황에 따라 여러 가지 알고리즘이 있음
  - SSSP(Single Source Shortest Path)
  - APSP(All Pair Shortest Path)
  - 그래프의 형태(방향 유무, DAG, 트리 등등)
  - 가중치의 범위(양수, 실수 전체 등등)

- 가장 범용적인 알고리즘 3(+1)가지를 다룸

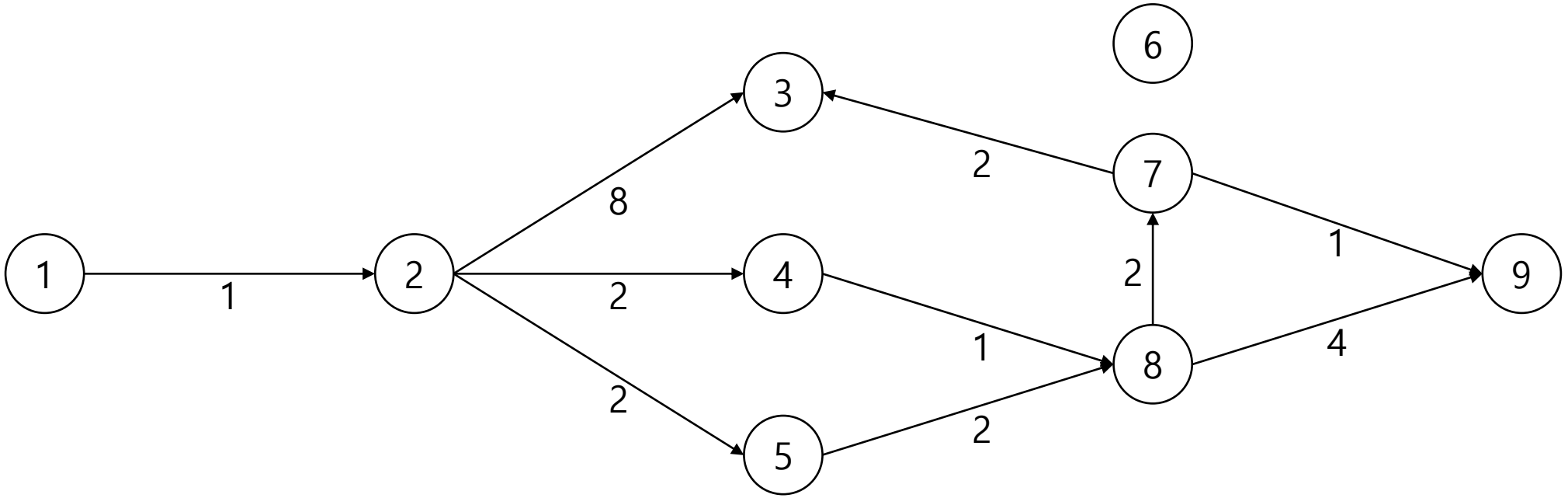
Dijkstra’s Algorithm

Dijktra's Algorithm

- 가중치가 0 이상인 그래프에서 SSSP를 푸는 알고리즘
- 시간 복잡도: O(V^2) / O(E log E)

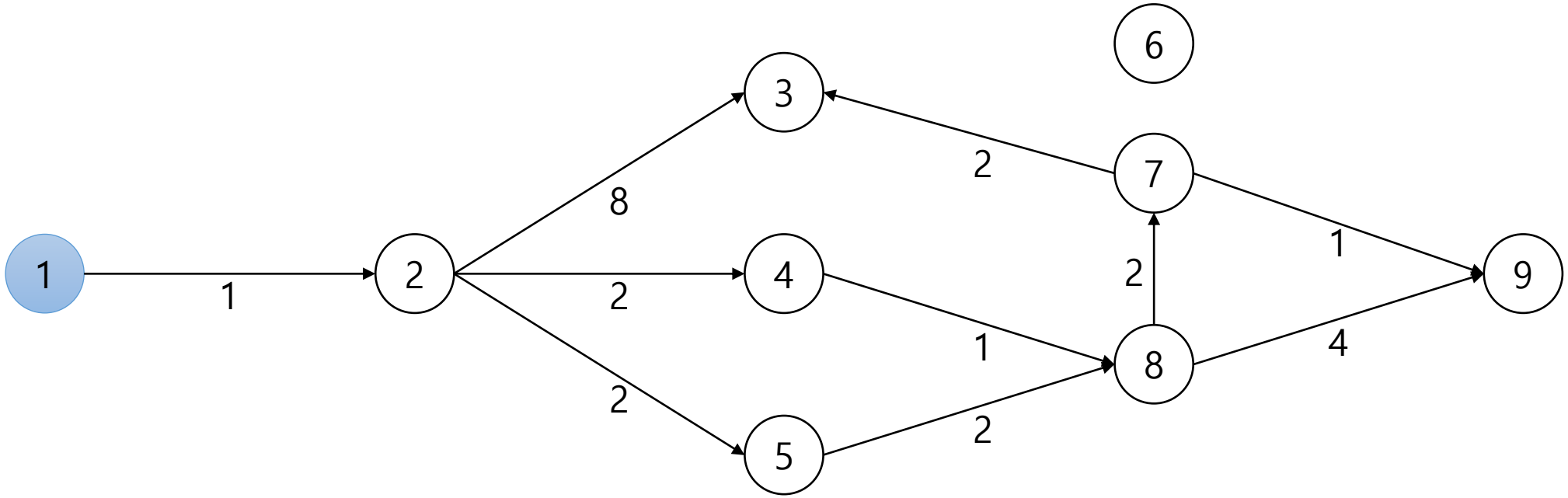
- 그리디 기반 알고리즘
  1. 시작점(S)까지의 거리는 0, 다른 모든 정점까지의 거리는 INF로 초기화
  2. 아직 거리가 "확정"되지 않은 정점 중 거리가 가장 짧은 정점(v) 선택 (처음에는 S를 선택함)
  3. v의 거리를 "확정"시킴
  4. v와 인접하면서 아직 거리가 "확정"되지 않은 정점들의 거리를 갱신(D[i] <- D[v] + weight)
  5. 모든 정점의 거리가 "확정"되었다면 종료 / 그렇지 않으면 2번으로 돌아감

Example (1)



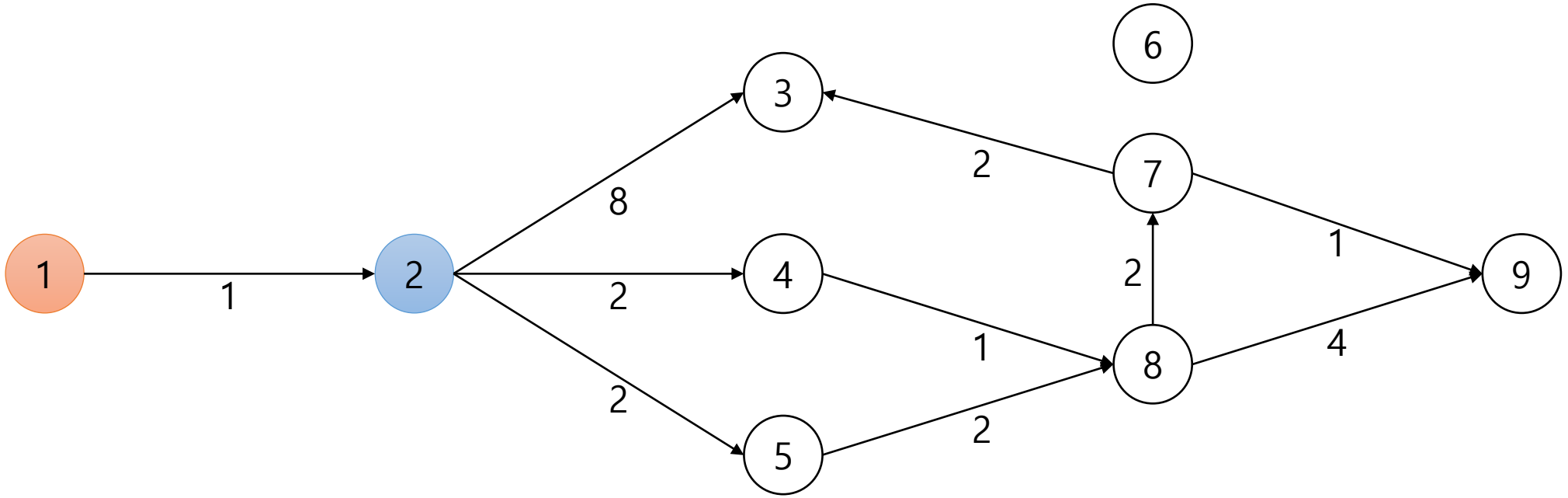
1	2	3	4	5	6	7	8	9
0	inf	inf	inf	inf	inf	inf	inf	inf

Example (2)



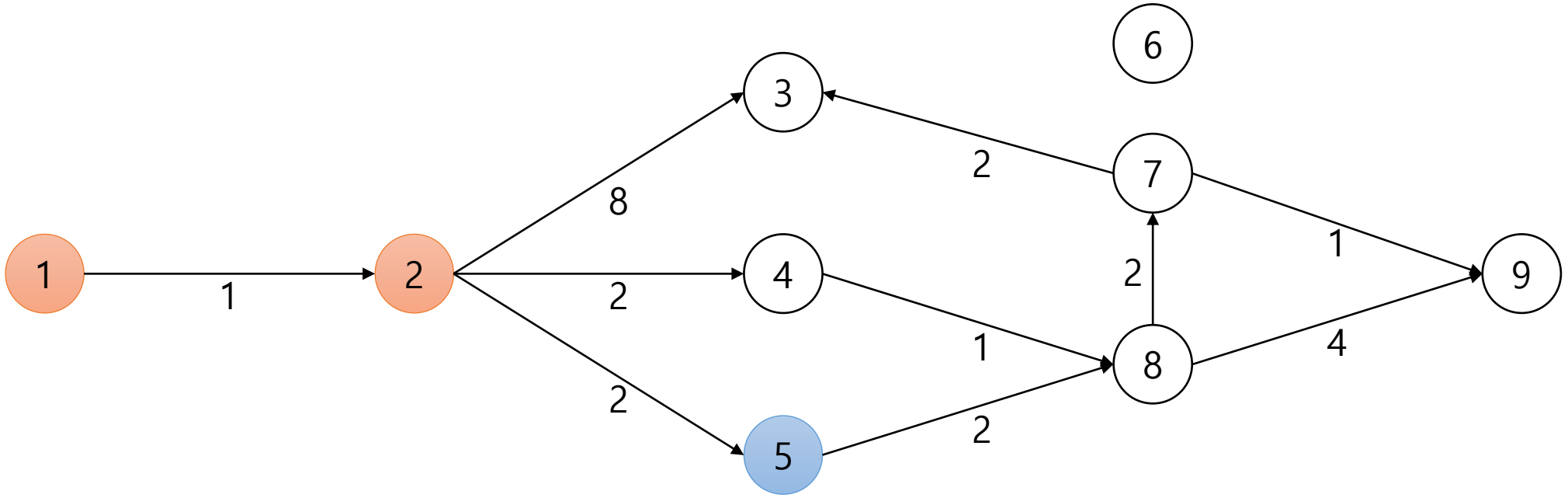
1	2	3	4	5	6	7	8	9
0	1	inf	inf	inf	inf	inf	inf	inf

Example (3)



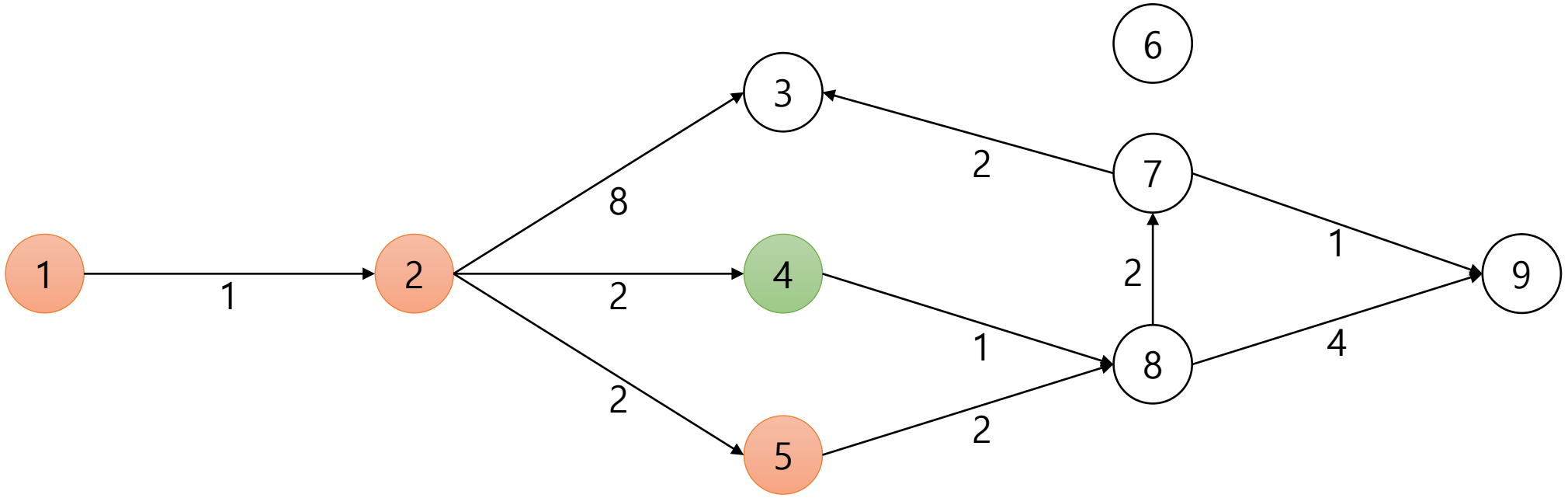
1	2	3	4	5	6	7	8	9
0	1	9	3	3	inf	inf	inf	inf

Example (4)



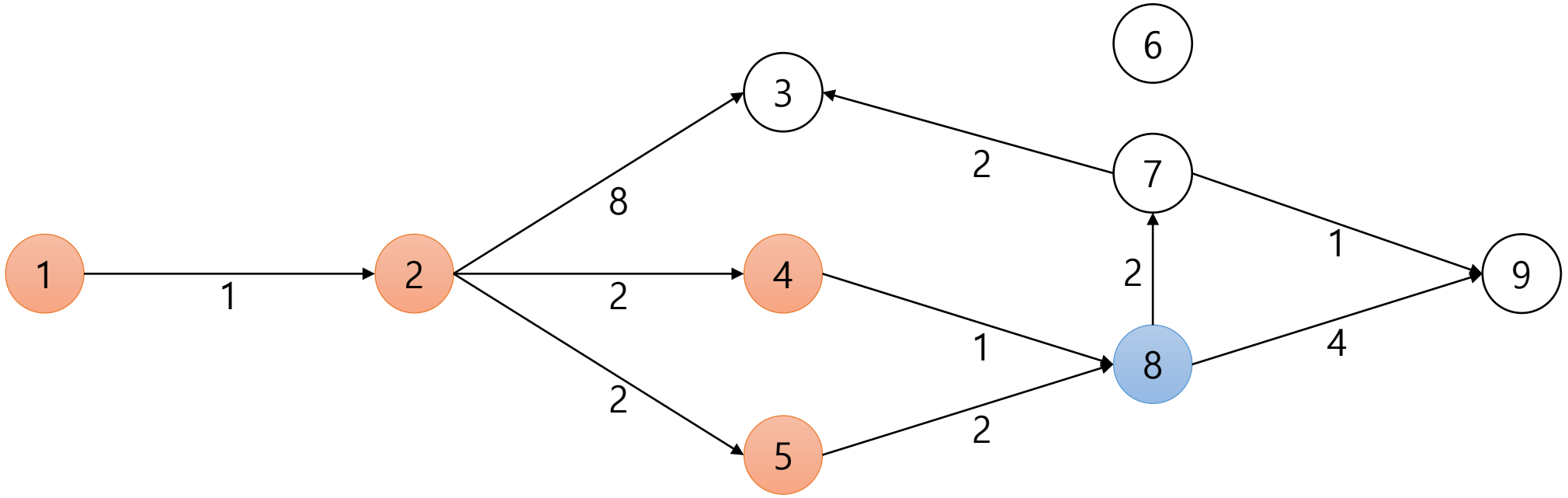
1	2	3	4	5	6	7	8	9
0	1	9	3	3	inf	inf	5	inf

Example (5)



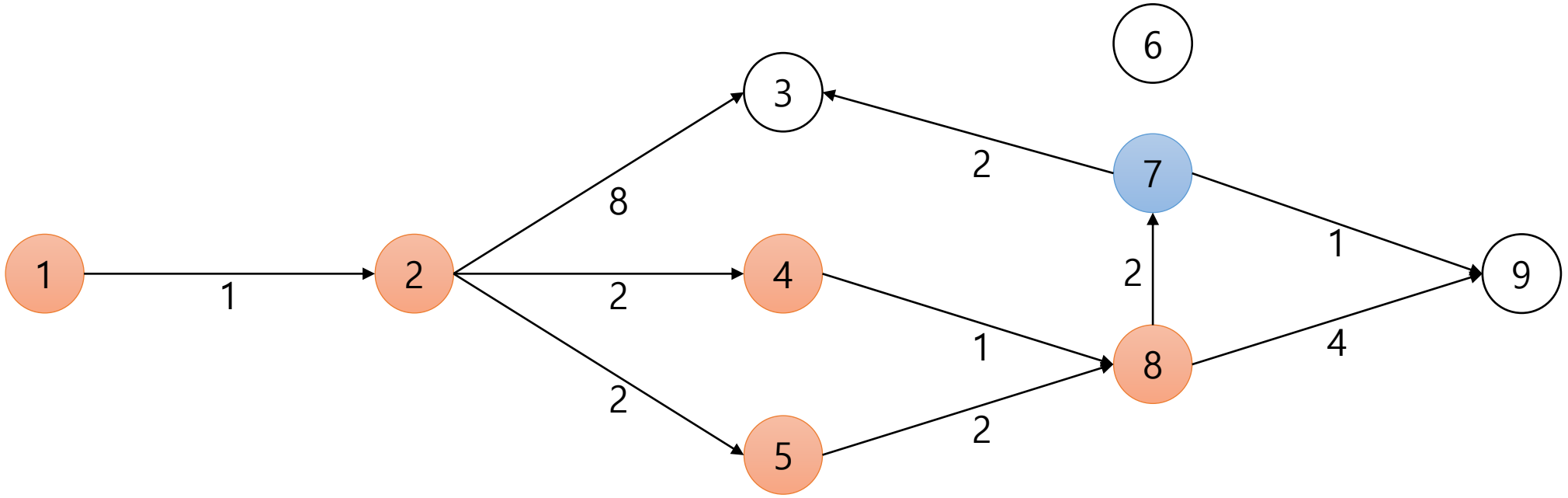
1	2	3	4	5	6	7	8	9
0	1	9	3	3	inf	inf	4	inf

Example (6)



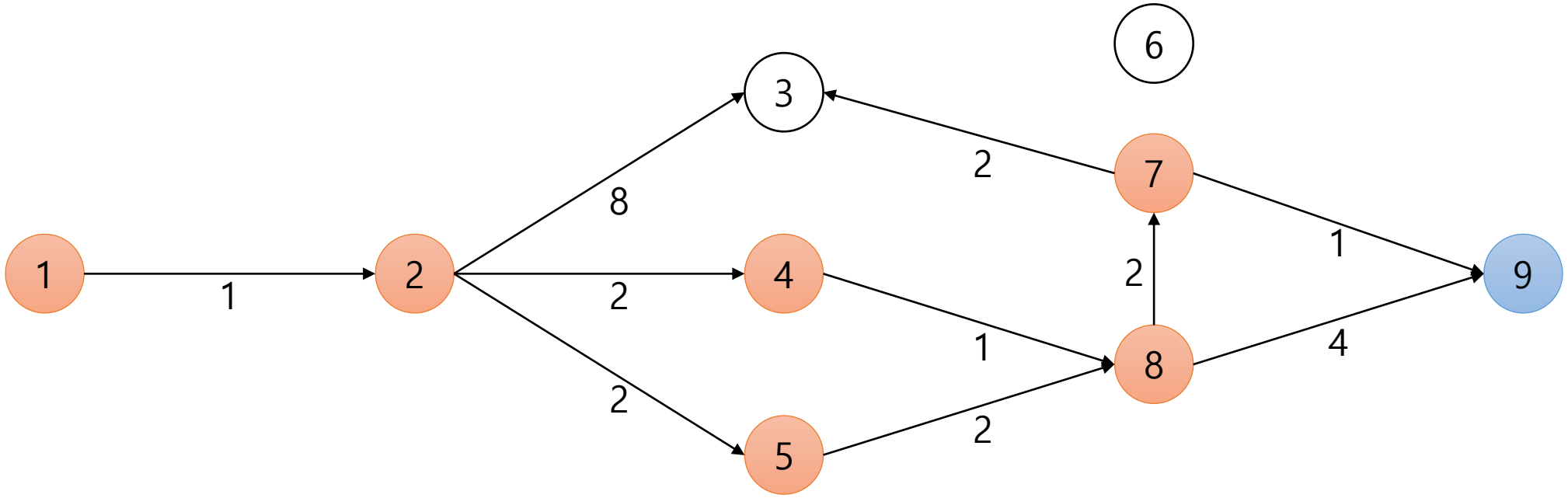
1	2	3	4	5	6	7	8	9
0	1	9	3	3	inf	6	4	8

Example (7)



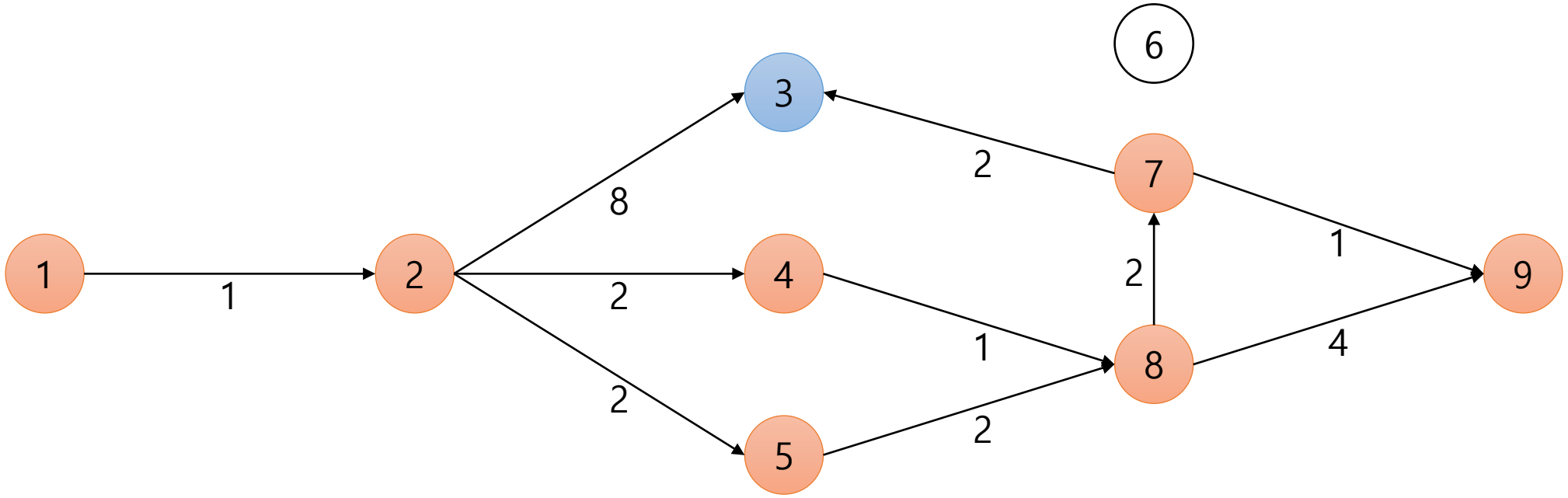
1	2	3	4	5	6	7	8	9
0	1	8	3	3	inf	6	4	7

Example (8)



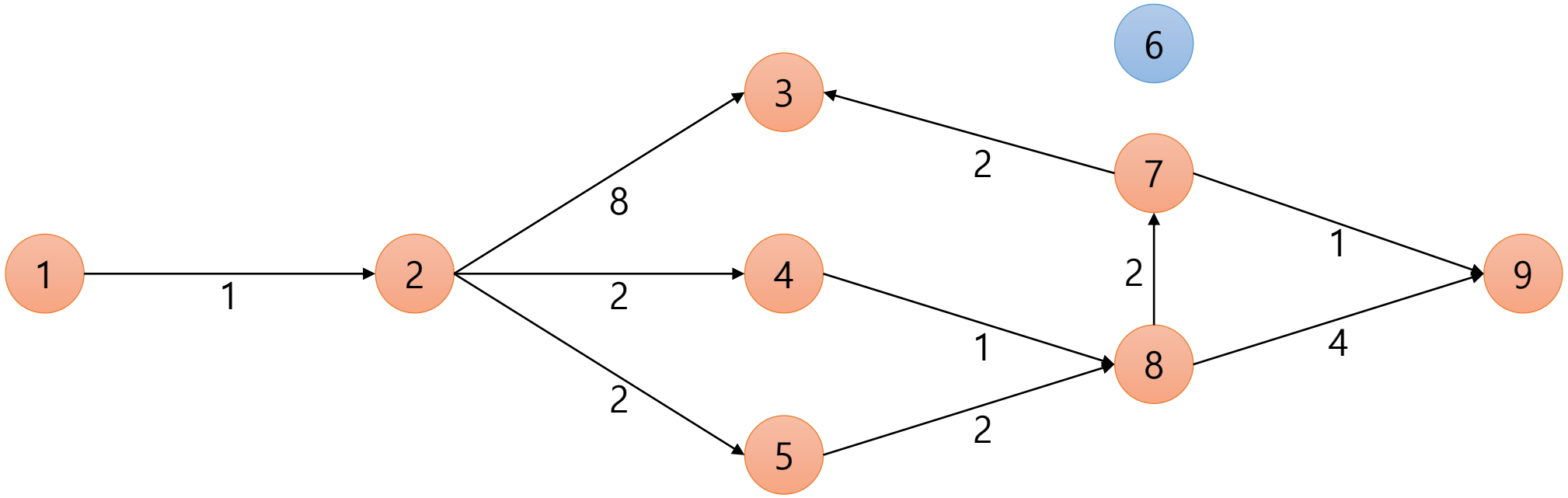
1	2	3	4	5	6	7	8	9
0	1	8	3	3	inf	6	4	7

Example (9)



1	2	3	4	5	6	7	8	9
0	1	8	3	3	inf	6	4	7

Example (10)



1	2	3	4	5	6	7	8	9
0	1	8	3	3	inf	6	4	7

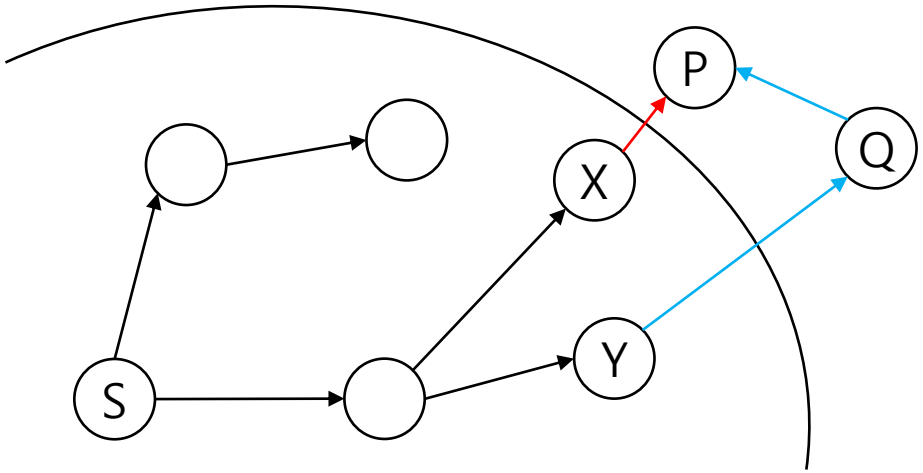
```cpp
using ll = long long;
using PLL = pair<ll, ll>;
const int MAX_N = 5050;

int N, M;
vector<PLL> G[MAX_N]; // 인접리스트
ll D[MAX_N];          // 거리
int C[MAX_N];         // 거리 확정 여부

void Dijkstra(int S){
    memset(D, 0x3f, sizeof D);                  // D 배열을 inf로 초기화
    D[S] = 0;                                   // 시작점의 거리는 0으로 초기화
    for(int iter=1; iter<=N; iter++){
        int v = -1;
        for(int i=1; i<=N; i++){
            if(C[i]) continue;                  // 이미 거리가 확정된 정점은 넘어감
            if(v == -1 || D[v] > D[i]) v = i; // 거리가 가장 짧은 정점 선택
        }
        if(v == -1) break;
        C[v] = 1;                               // 거리 확정
        for(auto i : G[v]){
            if(D[i.first] > D[v] + i.second){ // 거리 갱신
                D[i.first] = D[v] + i.second;
            }
        }
    }
}
```

Proof of Correctness

- 현재까지 거리가 확정된 정점 집합 U / 그 다음으로 거리가 가장 짧은 정점 P
- P까지의 최단 경로가 U에 있는 정점으로만 구성됨을 증명하면 됨
- 귀류법을 사용하자.
  - P까지의 최단 경로가 U 밖에 있는 정점 Q를 지난다고 하자.
  - Q에서 P로 가는 간선은 가중치가 0 이상이기 때문에
  - U에서 P로 가는 것보다 U에서 Q를 거쳐서 P로 가는 것이 더 짧다면
  - S에서 P까지의 거리보다 S에서 Q까지의 거리가 더 짧기 때문에 P를 선택하지 않게 된다.



질문?

Time Complexity

- V번의 iteration
  - 거리가 확정되지 않은 정점 중 최솟값 찾기 : O(V)
  - V에서 갈 수 있는 정점들의 거리 갱신 : O(deg(V))
- O(sum(V + deg(i))) = O(V^2 + E) = O(V^2)
  - Handshaking Lemma: sum(deg(i)) = 2E

- V에서 뻗어 나가는 간선은 모두 봐야 하므로 O(deg(V))가 하한임
- 거리가 최소인 정점을 빠르게 찾을 수 있을까?

- Heap!

Dijkstra's Algorithm with Heap

```cpp
using ll = long long;
using PLL = pair<ll, ll>;
const int MAX_N = 5050;

int N, M;
vector<PLL> G[MAX_N]; // 인접리스트
ll D[MAX_N];          // 거리
int C[MAX_N];         // 거리 확정 여부

void Dijkstra(int S){
    memset(D, 0x3f, sizeof D);                      // D 배열을 inf로 초기화
    priority_queue<PLL, vector<PLL>, greater<>> pq; // {거리, 정점} pair를 저장하는 min-heap
    D[S] = 0; pq.emplace(0, S);                     // 시작 정점 거리 초기화, {0, S}를 heap에 삽입
    while(!pq.empty()){
        // (Structured binding declaration 문법)
        auto [cst,now] = pq.top(); pq.pop();        // cst: now 까지의 거리
        if(C[now]) continue;                        // 이미 거리가 확정되었으면 넘어감
        C[now] = 1;                                 // now 까지의 거리 확정
        for(auto [nxt,len] : G[now]){               // {다음 정점, 간선 길이}
            if(D[nxt] > D[now] + len){              // 거리 갱신
                D[nxt] = D[now] + len;
                pq.emplace(D[nxt], nxt);
            }
        }
    }
}
```

Time Complexity

- for(auto [nxt,len] : G[now])의 총 반복 횟수는 O(E)
  - 각 정점마다 한 번씩 들어가므로, sum(deg(i))번 반복함
- 거리 갱신 최대 O(E)번 발생하므로 Heap에 원소 O(E)번 삽입
- Heap의 크기는 최대 O(E)이므로 시간 복잡도는 O(E log E)

- 참고
  - 거리 배열은 V * (간선 가중치 최댓값)으로 초기화 : 모든 경로는 최대 V-1개의 간선으로 구성
  - Heap에 각 정점마다 원소가 최대 한 개씩 존재하도록 구현하면 O(E log V)
  - Heap의 decrease key 연산을 O(1)에 구현하면 O(E + V log V)도 가능 (Fibonacci Heap)

Dijkstra's Algorithm with Thin Heap

- O(E + V log V)를 사용하고 싶다면 GCC Extensions를 사용하자

```cpp
#include <bits/extc++.h>
using namespace std;

using ll = long long;
using PLL = pair<ll, ll>;

int N, M;
ll D[500000];
vector<PLL> G[500000];

void Dijkstra(int S){
    __gnu_pbds::priority_queue<PLL, greater<>, __gnu_pbds::thin_heap_tag> pq;
    vector<decltype(pq)::point_iterator> iter(N);
    memset(D, 0x3f, sizeof D);
    for(int i=0; i<N; i++) iter[i] = pq.push(PLL(D[i], i));
    pq.modify(iter[S], PLL(D[S]=0, S));
    while(pq.size()){
        auto [cst,v] = pq.top(); pq.pop();
        if(cst != D[v]) continue;
        for(auto [i,c] : G[v]) if(D[i] > cst + c) pq.modify(iter[i], PLL(D[i]=cst+c, i));
    }
}
```

Applications

- 정점에 가중치가 있는 경우
  - 정점을 두 개로 분할 -> in(v), out(v)
  - u에서 v로 가는 간선 -> out(u)에서 in(v)로 가는 간선
  - 정점의 가중치 w(v) -> in(v)에서 out(v)로 가는 가중치 w(v) 간선

- {S1, S2, … , Sk}에서 다른 모든 정점으로 가는 최단 거리
  - Multi Source Shortest Path
  - 새로운 정점 S0에서 S1, S2, … , Sk로 가는 가중치 0 간선을 만들면
  - S0에서 다른 모든 정점으로 가는 최단 거리 문제로 바뀜 -> SSSP

질문?

Floyd-Warshall Algorithm

Floyd-Warshall Algorithm

- APAP를 푸는 알고리즘
- 시간 복잡도: O(N^3)
- 공간 복잡도: O(N^3) -> O(N^2)

- DP 기반 알고리즘
  - D[k][i][j]: i에서 출발해서 1..k번 정점을 경유한 뒤 j번 정점으로 가는 최단 거리
  - D[0][i][i]는 0으로 초기화
  - i ≠ j 일 때 D[0][i][j]는 간선이 있으면 간선의 가중치, 없으면 INF로 초기화
  - D[k][i][j] <- D[k-1][i][k] + D[k-1][k][j]
  - 배열 D의 크기는 N^3

Floyd-Warshall Algorithm

- 배열 D의 크기를 2N^2로 줄일 수 있다.
  - D[k][i][j]를 계산할 때 D[k-1][*][*]만 사용하므로
  - D[2][N][N] 선언하고 토글링하면 됨

- 배열 D의 크기를 N^2으로 줄일 수 있다.
  - 최솟값을 구하는 문제인데, 값이 매번 감소하기 때문에 그냥 덮어써도 됨
  - 시간 복잡도는 여전히 O(N^3)

Floyd-Warshall Algorithm

```cpp
int N, M, D[555][555];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M;
    memset(D, 0x3f, sizeof D);           // D[i][j]를 inf로 초기화
    for(int i=1; i<=N; i++) D[i][i] = 0; // i에서 i로 가는 비용은 0
    for(int i=1; i<=M; i++){
        int st, ed, cst;
        cin >> st >> ed >> cst;
        D[st][ed] = min(D[st][ed], cst);
    }
    // 여기까지 오면, D[i][j]는 어떠한 정점도 경유하지 않고 직접 i에서 j로 가는 비용이다.

    for(int k=1; k<=N; k++){
        for(int i=1; i<=N; i++){
            for(int j=1; j<=N; j++){
                D[i][j] = min(D[i][j], D[i][k] + D[k][j]);
            }
        }
        // D[i][j] : 1..k번 정점을 경유해서 i에서 j로 가는 최단 거리
    }
}
```

질문?

Bellman-Ford Algorithm

Bellman-Ford Algorithm

- SSSP를 푸는 알고리즘
- 시간 복잡도: O(VE)

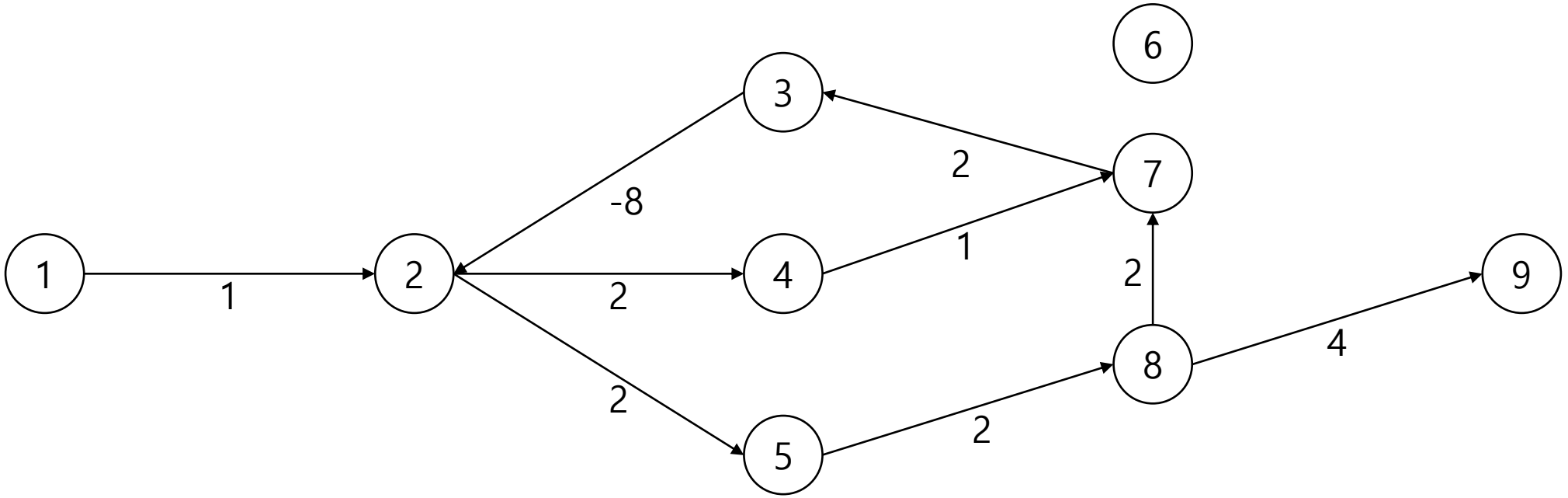
- 몇 가지 관찰을 해보자
  - 관찰1) 가중치의 합이 음수인 사이클이 있으면 최단 거리를 구할 수 없음 (음의 무한대로 발산)
  - 관찰2) 음수 사이클을 지나지 않는다면, 모든 최단 경로는 V-1개의 간선을 지남
- Dijkstra's Algorithm처럼 relaxation을 이용해 최단 거리를 구함
  - 시작 정점 S의 거리는 0, 다른 모든 정점까지의 거리는 INF로 초기화
  - 모든 간선 (s, e, w)에 대해 D[e] = min(D[e], D[s] + w)를 수행하는 과정을 V-1번 반복
    - i번째 iteration이 끝나면, 간선 i개를 거쳐서 가는 최단 거리를 정확하게 구할 수 있음
    - 귀납법으로 증명 가능함

Bellman-Ford Algorithm

- 음수 사이클이 존재하는지 판별하는 것도 가능하다!

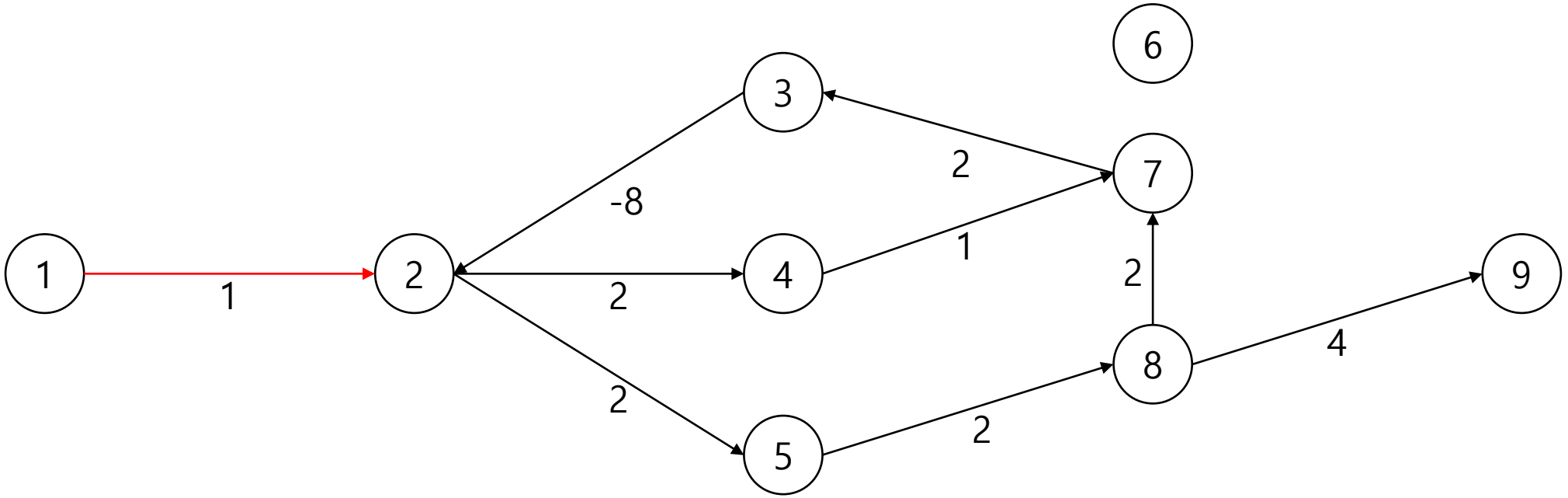
- 음수 사이클이 없다면 V-1번의 iteration으로 최단 거리를 구할 수 있음
- 만약 V번째 iteration에서 relaxation이 발생한다면? -> 음수 사이클 존재

Example (0)



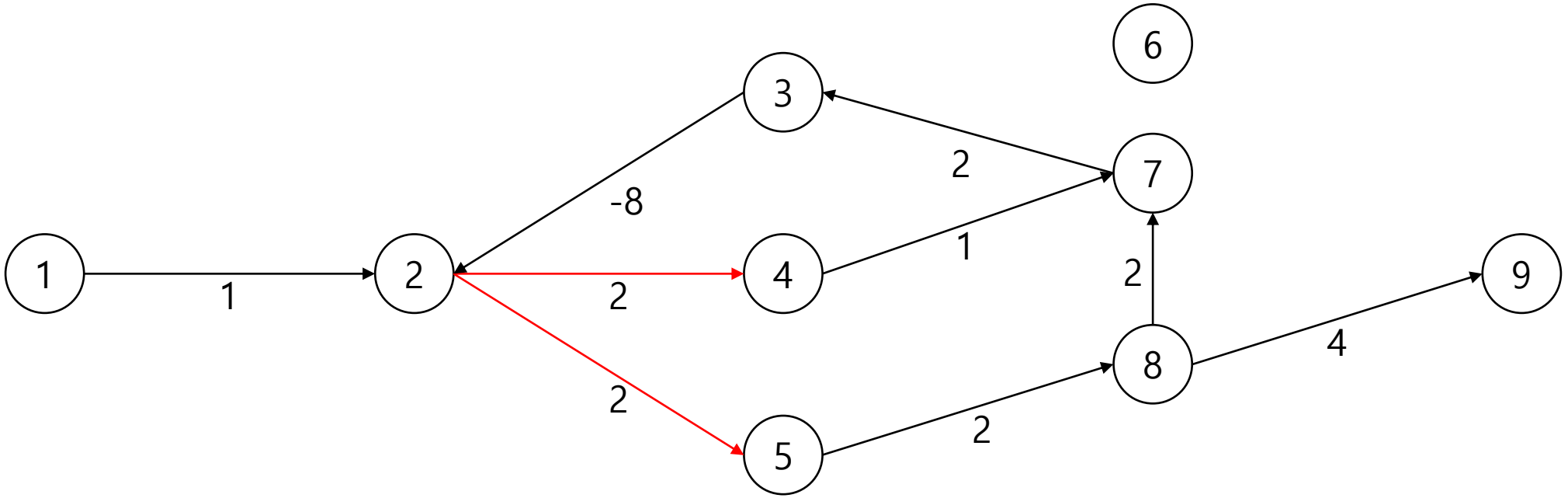
1	2	3	4	5	6	7	8	9
0	inf	inf	inf	inf	inf	inf	inf	inf

Example (1)



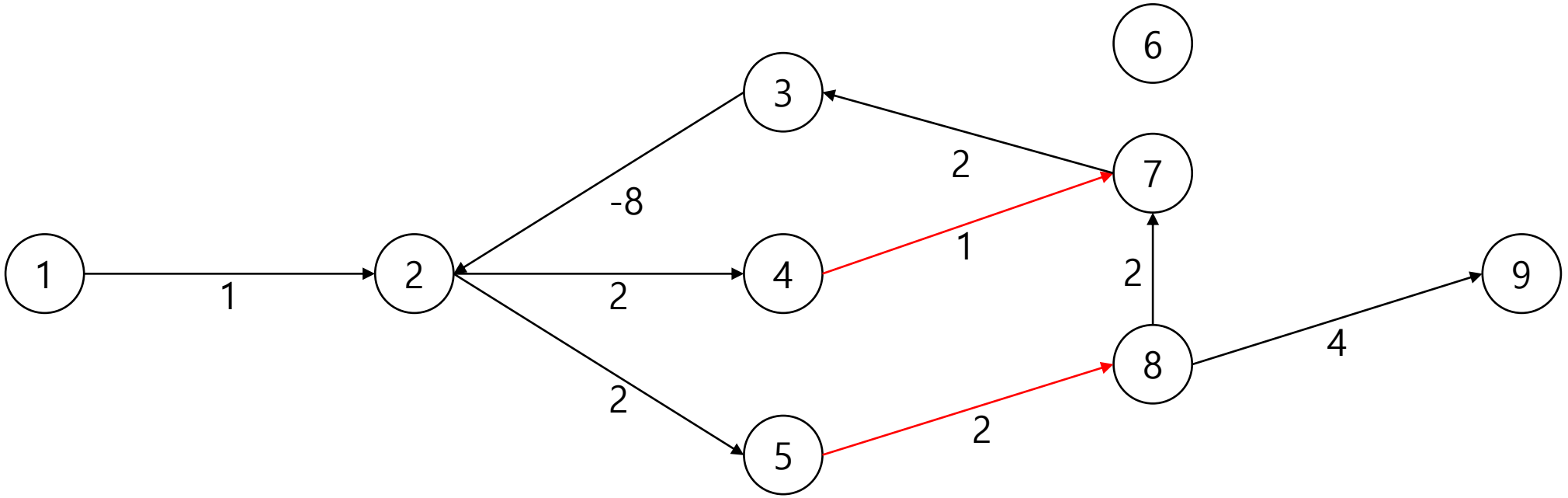
1	2	3	4	5	6	7	8	9
0	1	inf	inf	inf	inf	inf	inf	inf

Example (2)



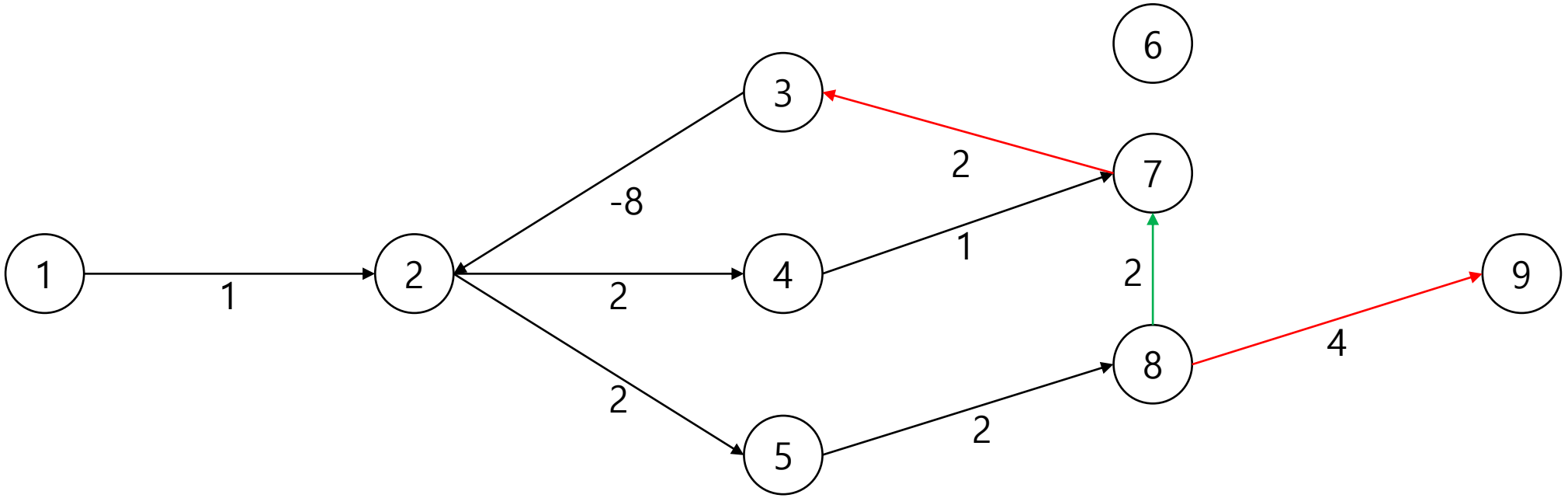
1	2	3	4	5	6	7	8	9
0	1	inf	3	3	inf	inf	inf	inf

Example (3)



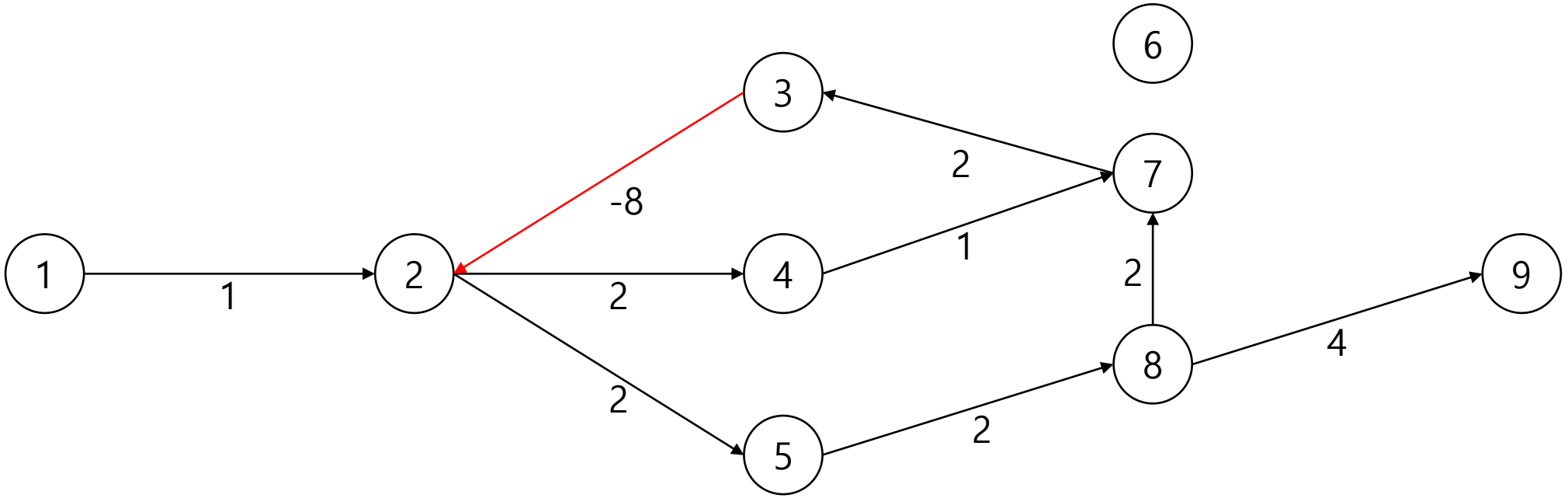
1	2	3	4	5	6	7	8	9
0	1	inf	3	3	inf	4	5	inf

Example (4)



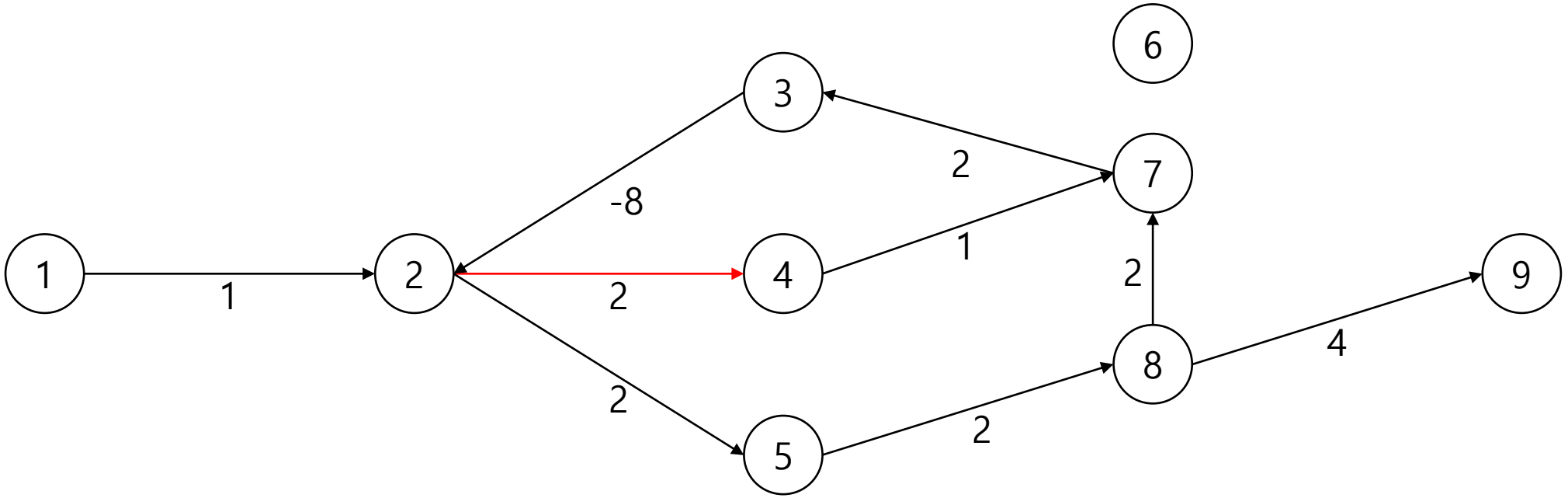
1	2	3	4	5	6	7	8	9
0	1	6	3	3	inf	4	5	9

Example (5)



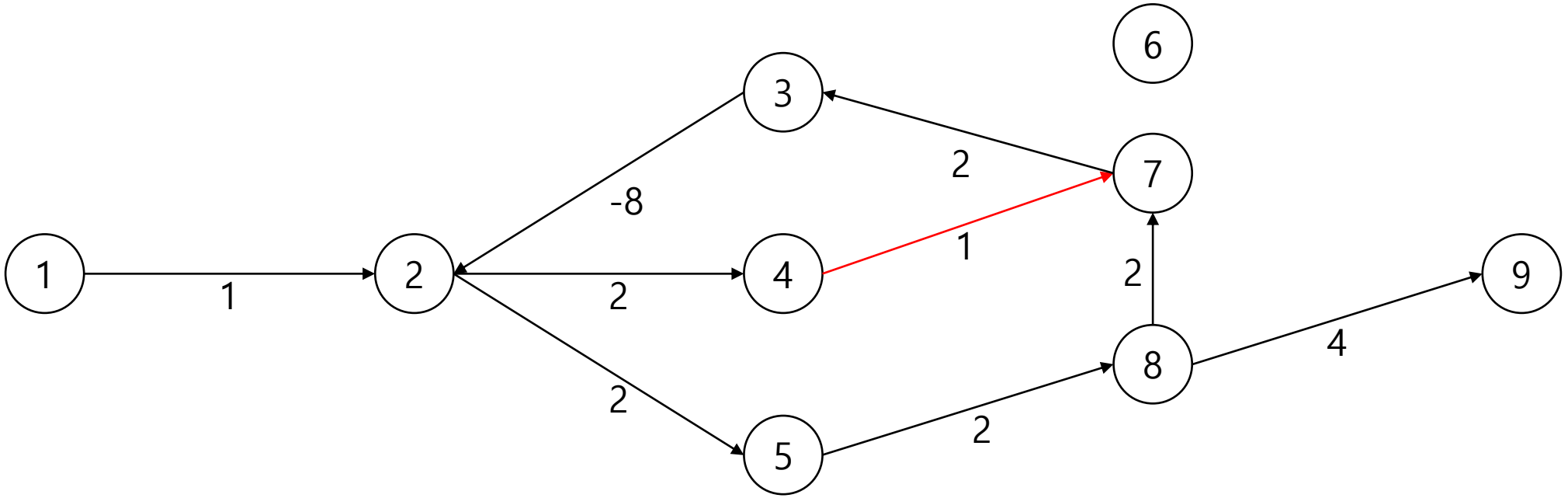
1	2	3	4	5	6	7	8	9
0	-2	6	3	3	inf	4	5	9

Example (6)



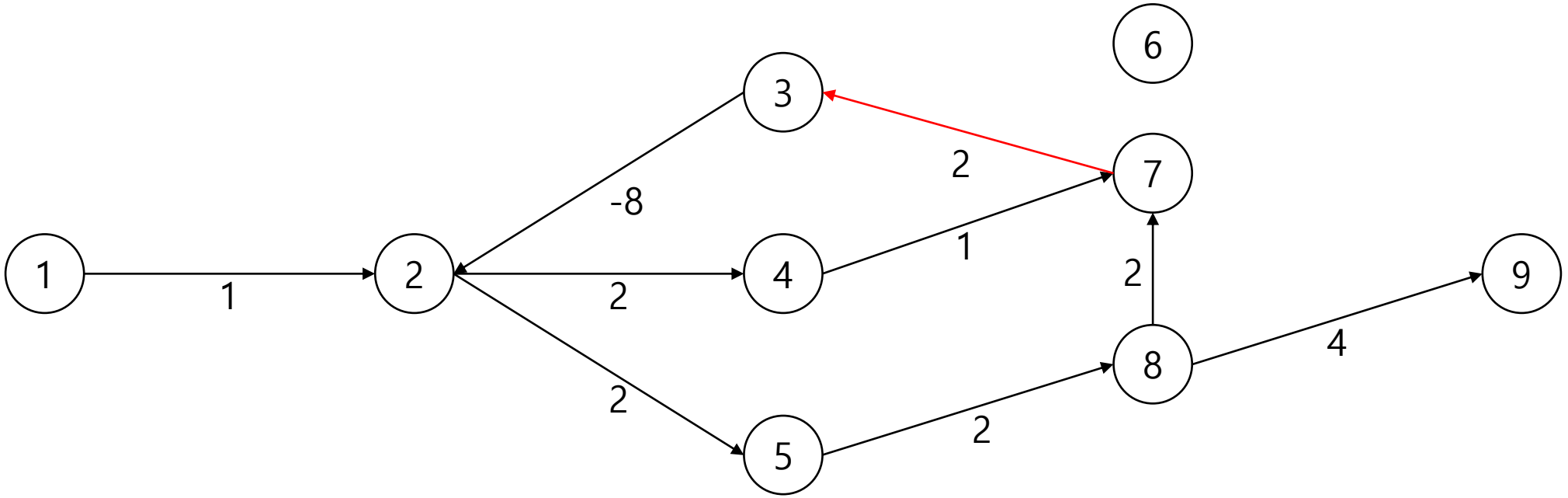
1	2	3	4	5	6	7	8	9
0	-2	6	0	3	inf	4	5	9

Example (7)



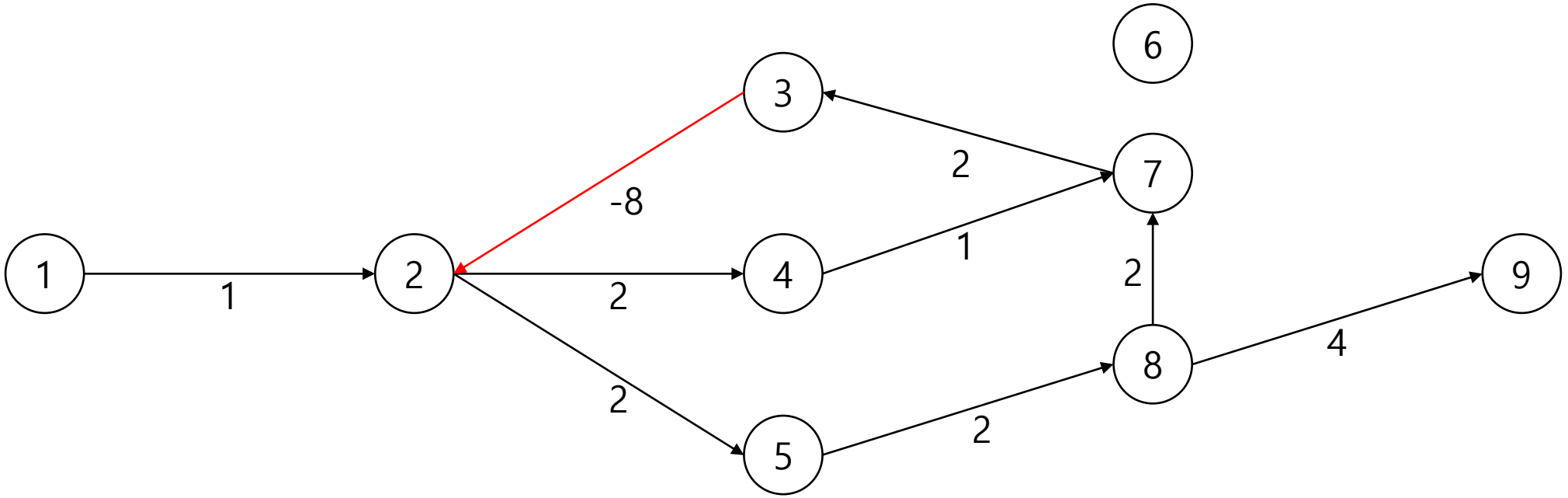
1	2	3	4	5	6	7	8	9
0	-2	6	0	3	inf	1	5	9

Example (8)



1	2	3	4	5	6	7	8	9
0	-2	3	0	3	inf	1	5	9

Example (9)



1	2	3	4	5	6	7	8	9
0	-5	3	0	3	inf	1	5	9

Bellman-Ford Algorithm

```cpp
using ll = long long;
constexpr ll INF = 0x3f3f3f3f3f3f3f3f;

struct Edge{
    int s, e, w;
    Edge() = default;
    Edge(int s, int e, int w) : s(s), e(e), w(w) {}
};

int N, M;
vector<Edge> E;
ll D[555];

bool BellmanFord(int S){
    memset(D, 0x3f, sizeof D);
    D[S] = 0;
    for(int iter=1; iter<=N; iter++){
        bool isChanged = false;
        for(const auto &[s,e,w] : E){
            if(D[s] == INF) continue; // 간선의 출발점이 INF라면 넘어감
            if(D[e] > D[s] + w){      // relaxation
                D[e] = D[s] + w;
                isChanged = true;
            }
        }
        if(isChanged && iter == N){   // N번째 iteration에서 relaxation이 발생하면 음수 사이클 존재
            return false;
        }
    }
    return true;
}
```

Time Complexity

- V번의 iteration * E개의 간선 탐색 -> O(VE)

- 주의 사항
  - 거리 배열은 V * (간선 가중치 최댓값)으로 초기화 : 모든 경로는 최대 V-1개의 간선으로 구성
  - 실행 도중에 값이 V * E * (간선 가중치 최솟값)까지 내려갈 수 있음 : overflow 주의

  - E개의 간선이 (1,2,-x), (2,1,-x), (1,2,-x), (2,1,-x), … 순으로 주어진다고 하자.
  - 간선에 대한 반복문에서
    - 첫 번째 반복에서 D[2] = -x, 두 번째 반복에서 D[1] = -2x, 세 번째 반복에서 D[2] = -3x
    - E번째 반복에서 D[1] = D[2] ~= -xE/2
  - 이 과정을 V번 반복하면 D[1] = D[2] ~= -xVE/2

Applications

- $X_j - X_i \leq w(i, j)$ 꼴의 부등식이 여러 개 주어졌을 때 X_i 값을 구하는 문제
  - 가상의 정점 S에서 1, 2, …, N으로 가는 가중치 0 간선 만들고
  - i에서 j로 가는 가중치 w(i, j) 간선 만들면
  - S에서 시작하는 SSSP를 이용해 부등식을 만족하는 X_i를 구할 수 있음

- Minimum Cost Flow
  - 생략

질문?

Shortest Path Faster Algorithm (SPFA)

SPFA

- SSSP를 푸는 알고리즘: Bellman-Ford의 연산량을 줄인 알고리즘
- 시간 복잡도: O(VE)
- Bellman-Ford가 O(VE)인 이유: 매번 모든 간선을 다 보기 때문
  - 필요한 간선만 보는 방식으로 연산량을 줄일 수 있을 것 같다!
  - 바로 직전 iteration에서 간선의 시작점이 갱신된 경우에만 해당 간선을 고려하는 방식

  - 최악의 경우에는 O(VE)로 동일하지만, 보통 Bellman-Ford보다 빠르다!
  - 출제자가 데이터를 대충 만들었을 경우(랜덤 데이터) 평균적으로 O(V+E)

SPFA

```cpp
using ll = long long;
using PLL = pair<ll, ll>;

int N, M;
vector<PLL> G[555];
ll D[555];
bool InQ[555];

bool SPFA(int S){
    queue<int> Q;
    memset(D, 0x3f, sizeof D);
    memset(InQ, false, sizeof InQ);

    Q.push(S); D[S] = 0; InQ[S] = true;
    for(int iter=1; !Q.empty(); iter++){ // iteration
        if(iter > N) return false;       // iteration이 N을 넘어갔으므로 음수 사이클 존재
        int sz = Q.size();               // 현재 iteration에서 봐야 하는 정점의 개수
        while(sz--){
            int v = Q.front(); Q.pop();
            InQ[v] = false;
            for(auto [i,c] : G[v]){
                if(D[i] > D[v] + c){     // relaxation
                    D[i] = D[v] + c;
                    if(!InQ[i]){         // 만약 i가 Queue에 없다면 push
                        Q.push(i);
                        InQ[i] = true;
                    }
                }
            }
        }
    }
    return true;
}
```

Time Complexity

- 최악의 경우
  - 최대 O(V)번의 iteration
  - 매번 모든 정점이 relaxation된다면 모든 간선을 다 고려해야 하므로 O(VE)

- 랜덤 데이터에서 O(V+E)라고 하던데 증명은 모름...
  - 그래프의 간선 구성이 매번 달라지는 경우 SPFA를 쓰면 매우 빠르게 동작함
  - Ex) Minimum Cost Flow

질문?

Minimum Spanning Tree

용어 정의

- 부분 그래프(Subgraph) : 그래프의 정점과 간선의 일부를 선택해서 만든 그래프
- 신장 부분 그래프(Spanning ~) : 그래프의 모든 정점을 포함하는 부분 그래프
- 신장 포레스트(Spanning Forest) : 사이클이 없는 신장 부분 그래프
- 신장 트리(Spanning Tree) : 모든 정점이 연결된 신장 포레스트
- 최소 비용 신장 트리(Minimum ~) : 간선의 가중치의 합이 최소인 신장 트리

최소 신장 트리 알고리즘

- 그래프가 주어지면 최소 신장 트리를 구하는 알고리즘
- 여러 가지 알고리즘이 있다.
  - Prim's Algorithm (V^2, E log V, E + V log V 등등)
  - Kruskal's Algorithm (E log E)
  - Boruvka's Algorithm (E log V, 이건 설명 안 함)

Prim’s Algorithm

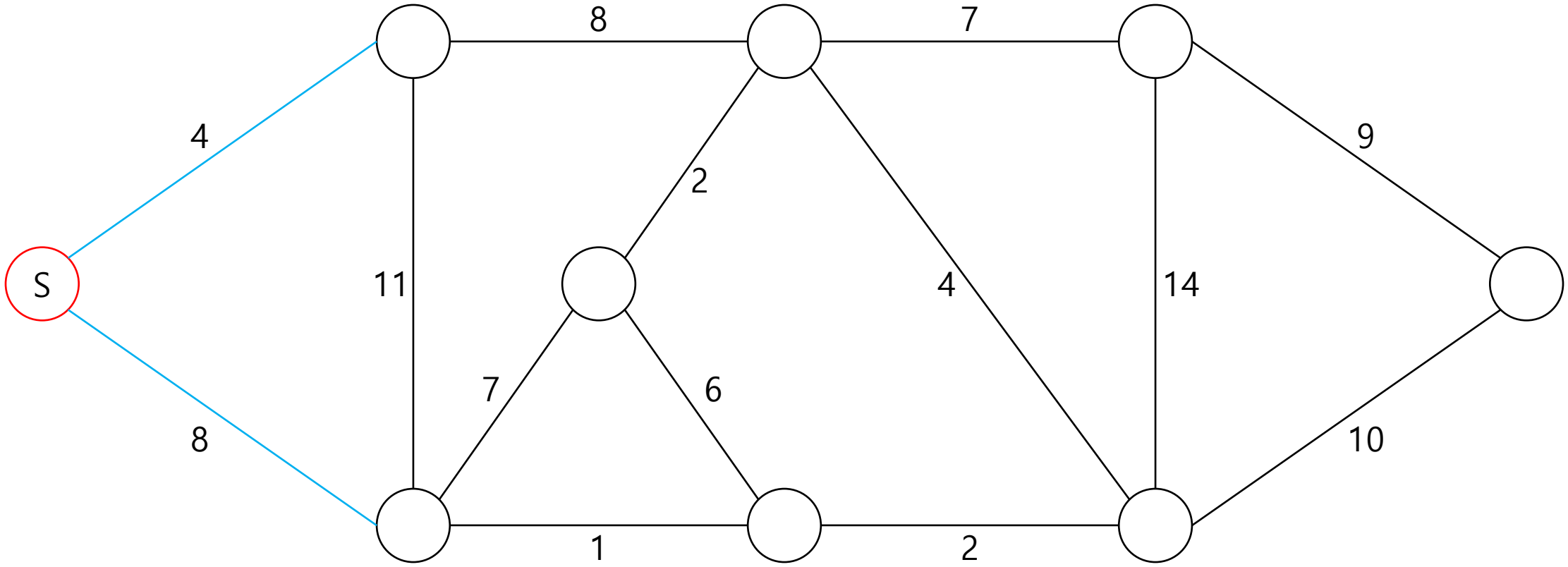
Prim's Algorithm

- Dijkstra's Algorithm과 매우 유사함
- 시간 복잡도: O(V^2) / O(E log E)

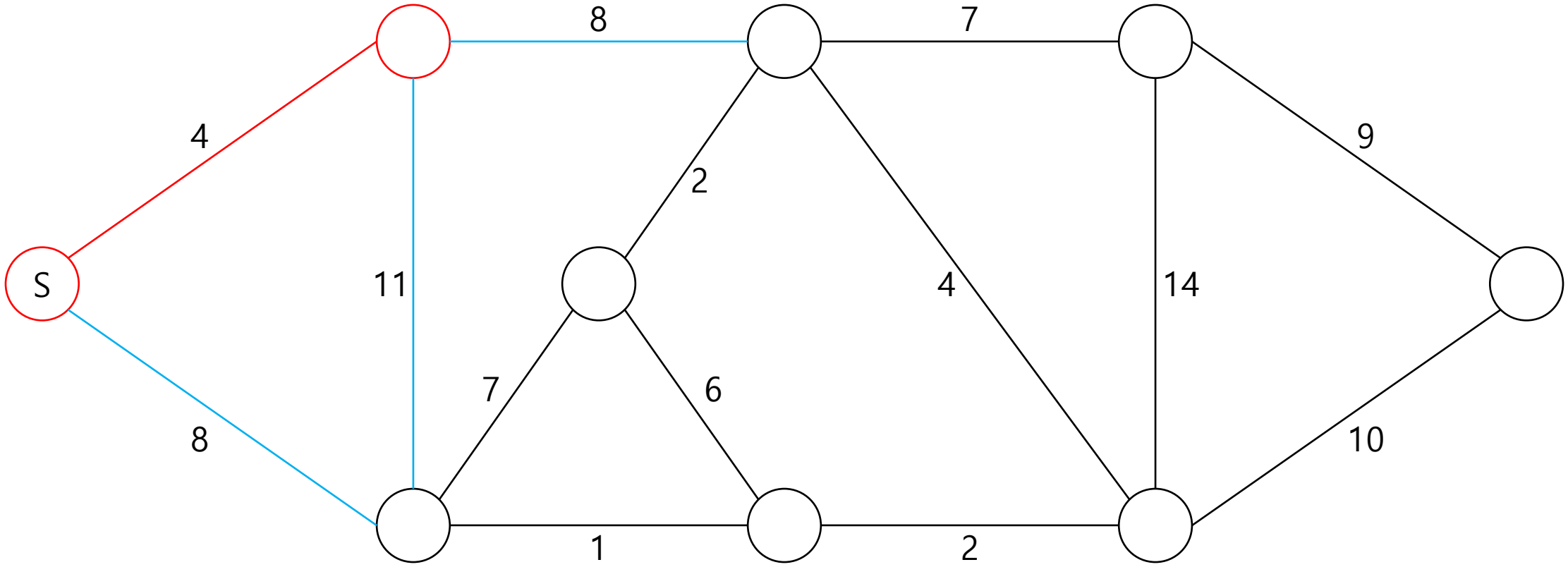
- Spanning Tree에 정점을 하나씩 포함시키면서 확장하는 방식으로 진행

- 그리디 기반 알고리즘
  1. 시작점(S)을 MST에 넣음
  2. 현재 MST에 있는 정점에서 뻗어 나가는 간선 중 가중치가 가장 작은 간선(e) 선택
  3. 만약 MST에 e를 추가할 수 있다면(사이클이 생기지 않는다면) e를 MST에 추가
     - 사이클 판별은 e = (u, v)에서 u와 v가 이미 MST에 포함되었는지 확인하면 됨

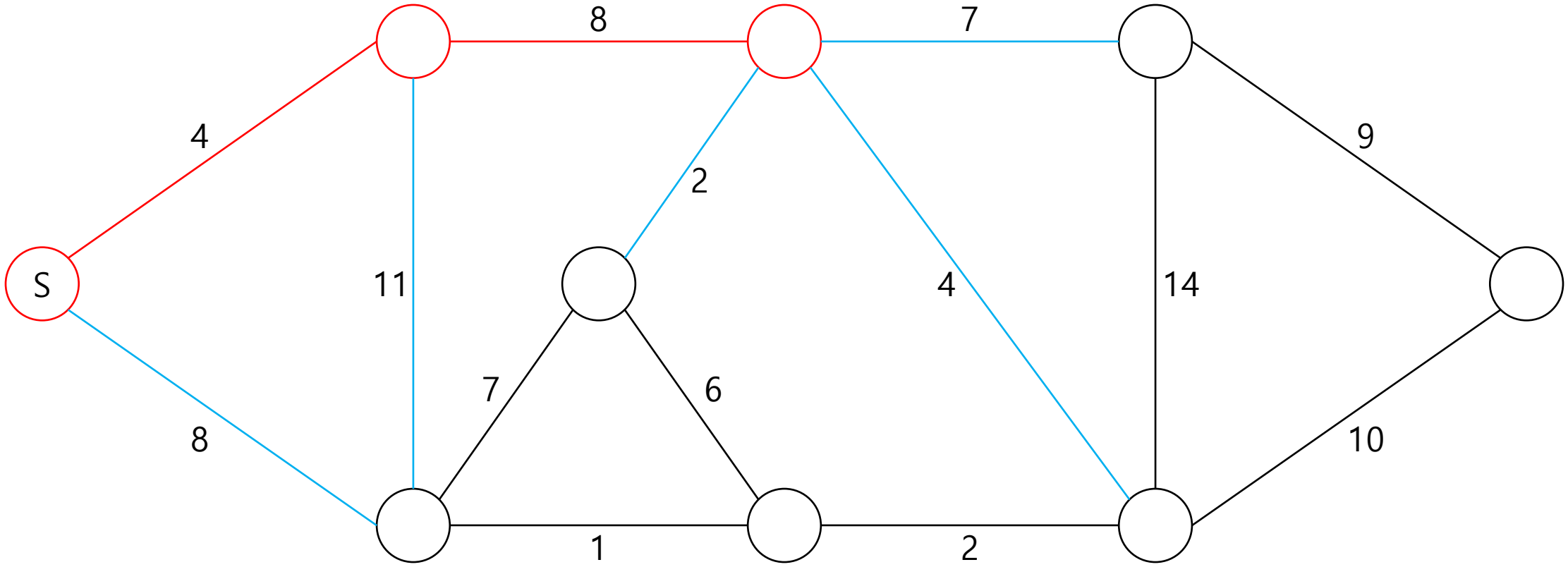
Example (1)



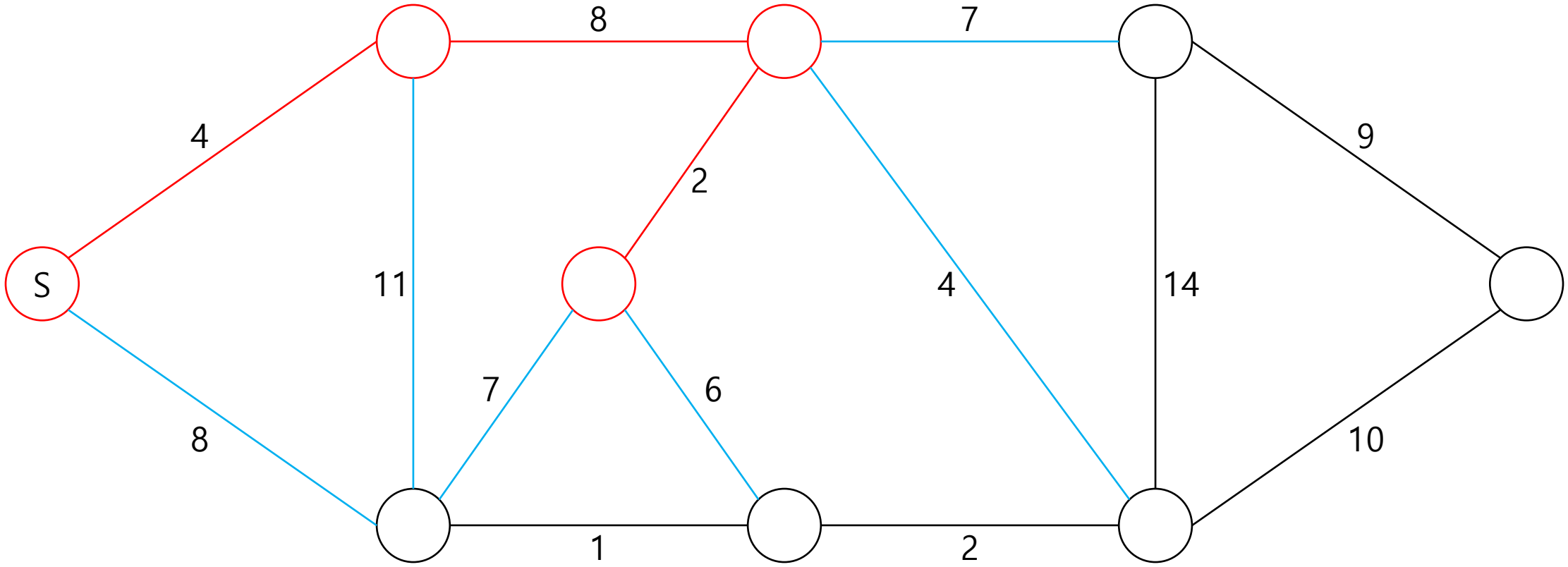
Example (2)



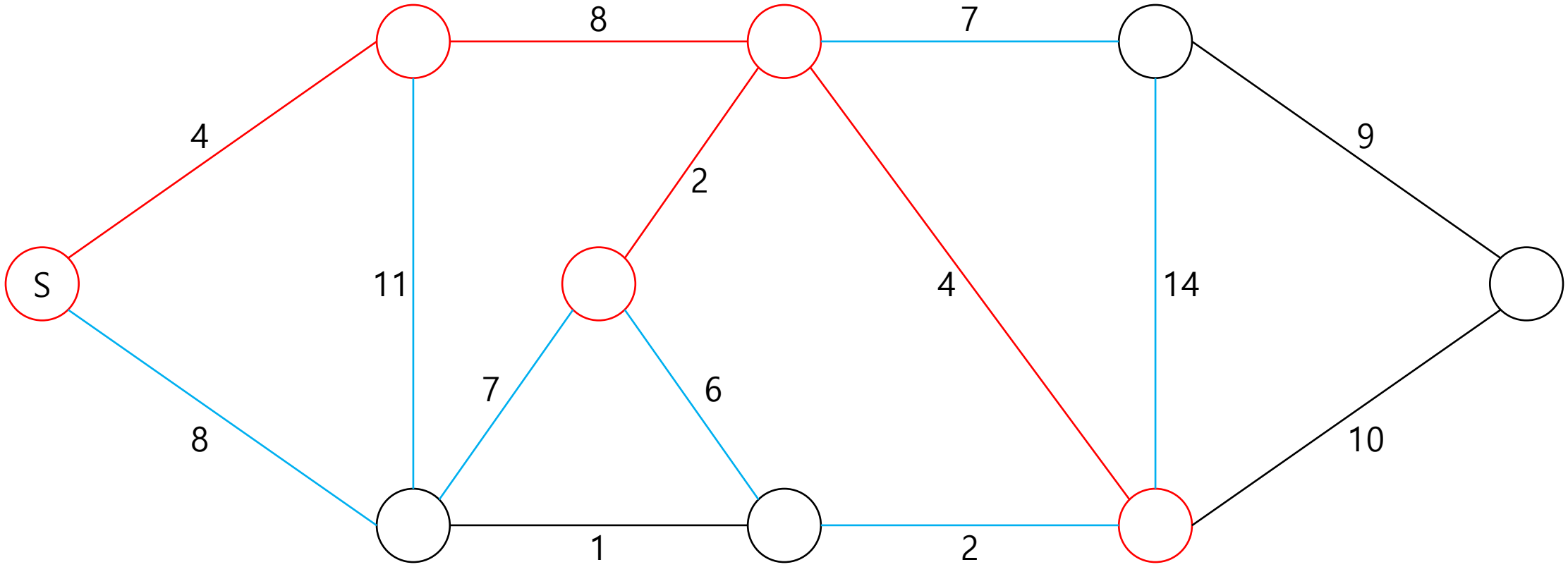
Example (3)



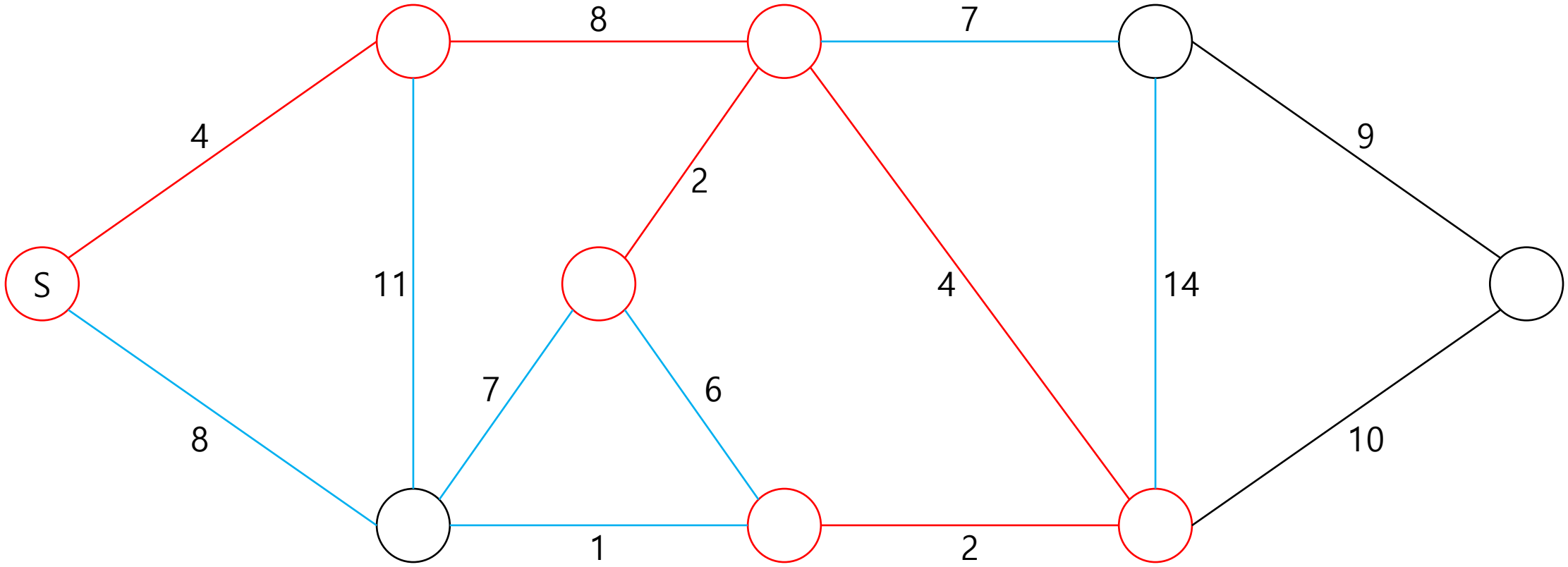
Example (4)



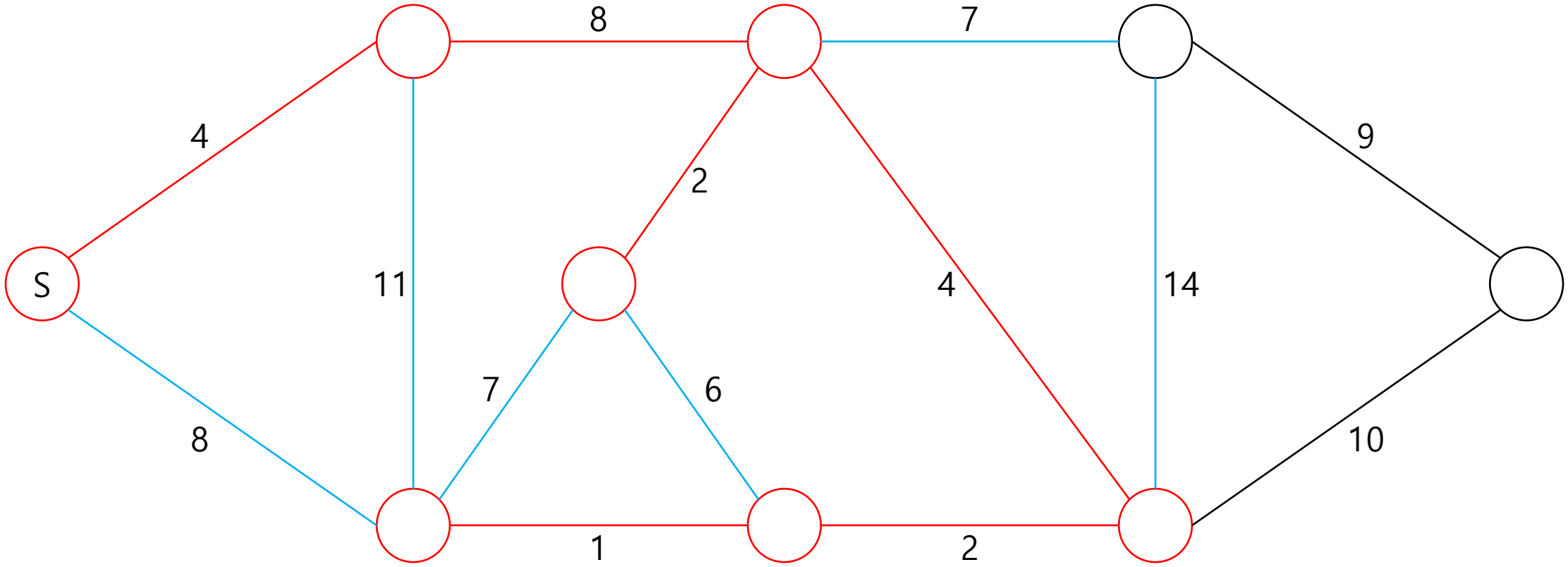
Example (5)



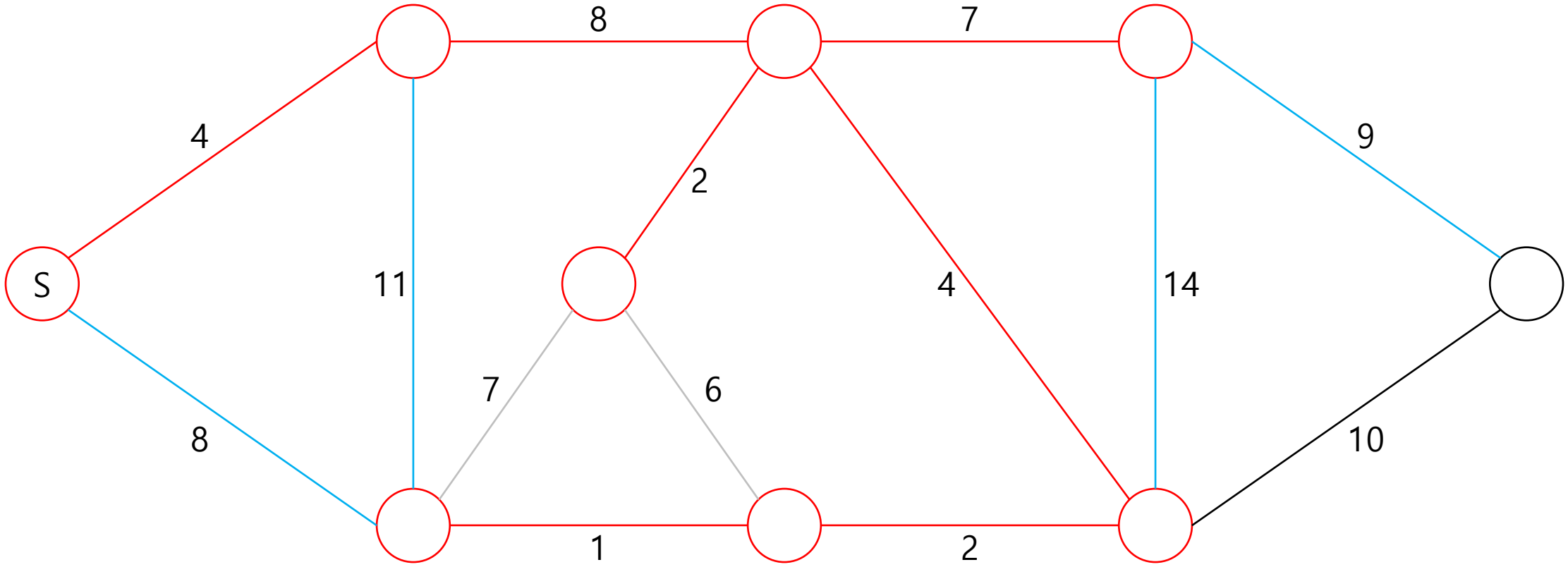
Example (6)



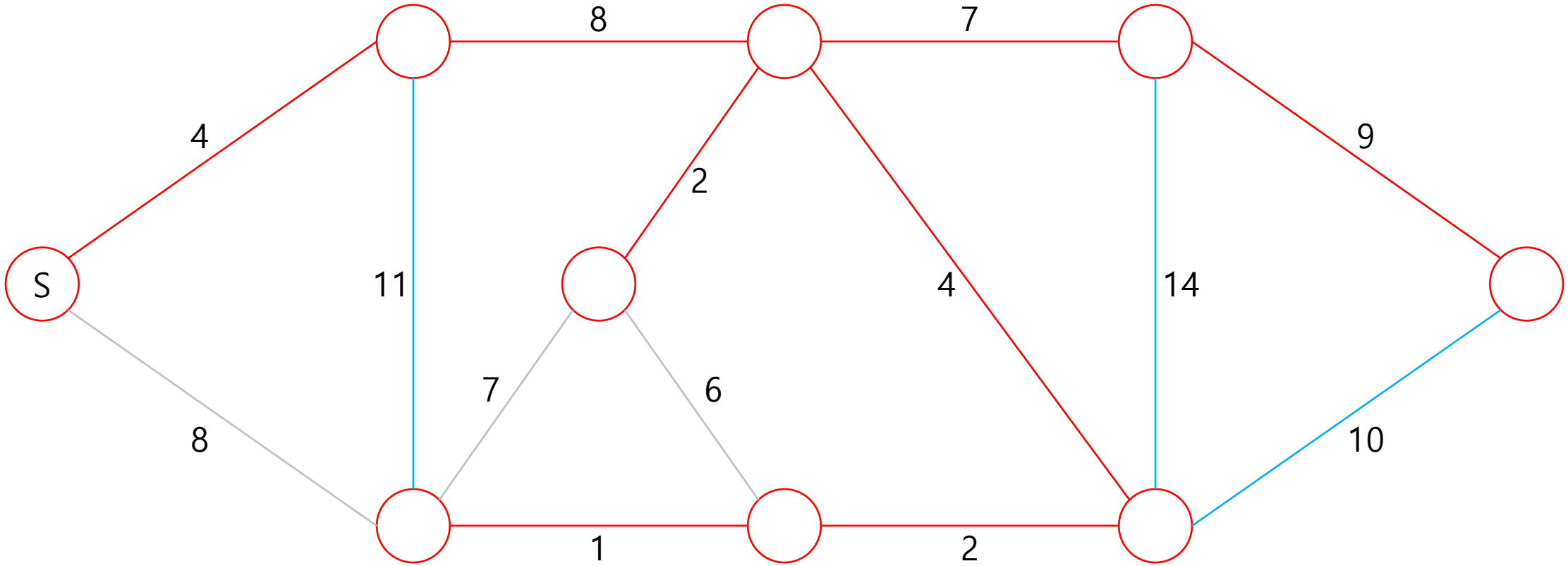
Example (7)



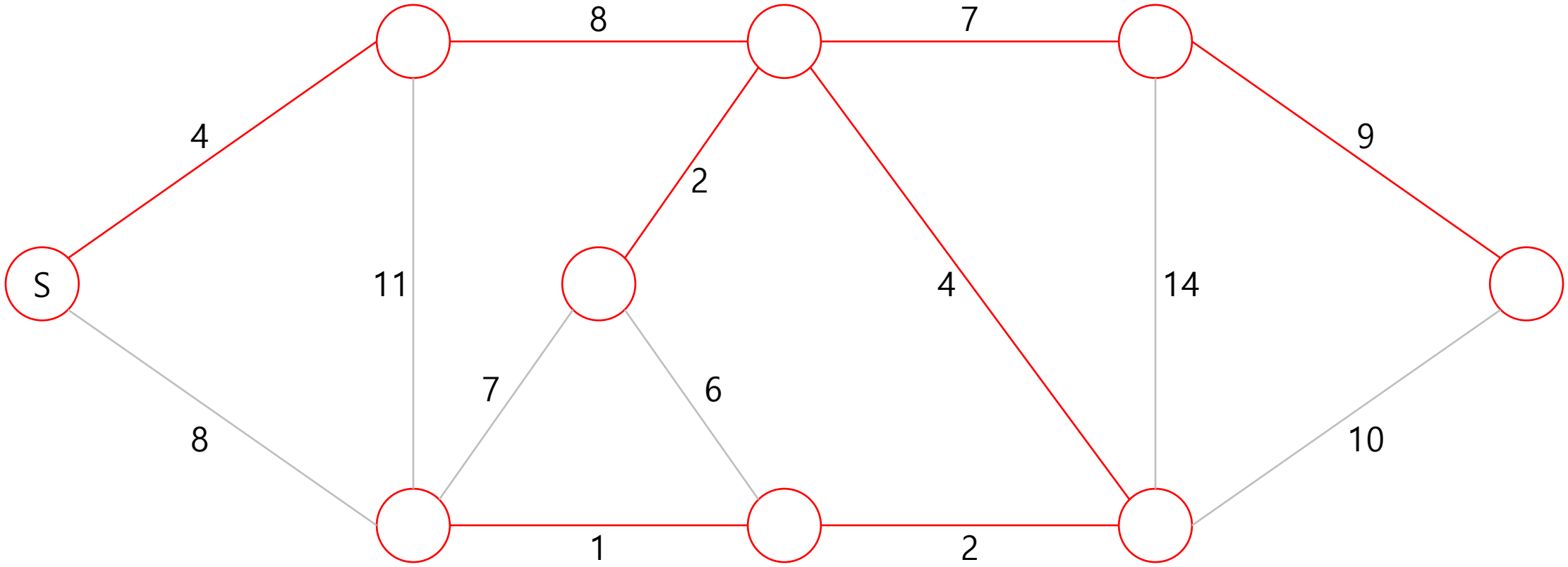
Example (8)



Example (9)



Example (10)



Prim's Algorithm

```cpp
using PII = pair<int, int>;
int N, M, C[5050], D[5050];
vector<PII> G[5050];

int Prim(){
    int ret = 0;
    memset(D, 0x3f, sizeof D);        // D[i] : i번 정점을 MST에 추가하기 위해 필요한 비용(간선 가중치)
    D[1] = 0;                         // 시작 정점 S = 1
    for(int iter=1; iter<=N; iter++){
        int v = -1;                   // 아직 MST에 포함되지 않은 정점 중 비용이 최소인 정점 선택
        for(int i=1; i<=N; i++){
            if(C[i]) continue;        // 이미 MST에 포함된 정점이라면 넘어감
            if(v == -1 || D[v] > D[i]) v = i;
        }
        C[v] = 1; ret += D[v];        // MST에 넣음
        for(auto [i,w] : G[v]){       // v에서 뻗어나가는 간선 정보 반영
            D[i] = min(D[i], w);
        }
    }
    return ret;
}
```

Proof of Correctness

- Dijkstra's Algorithm과 비슷한 방식으로 할 수 있다.
- 직접 해보자.

질문?

Time Complexity

- V번의 iteration
  - MST에 포함되지 않은 정점 중 비용이 최소인 정점 찾기 : O(V)
  - V에서 갈 수 있는 정점들의 거리 갱신 : O(deg(V))
- O(sum(V + deg(i))) = O(V^2 + E) = O(V^2)
  - Handshaking Lemma: sum(deg(i)) = 2E

- V에서 뻗어 나가는 간선은 모두 봐야 하므로 O(deg(V))가 하한임
- 비용이 최소인 정점을 빠르게 찾을 수 있을까?

- Heap!

Prim's Algorithm with Heap

```cpp
using PII = pair<int, int>;

int N, M, C[10101];
vector<PII> G[10101];

int Prim(){
    int ret = 0;
    priority_queue<PII, vector<PII>, greater<>> pq; // {거리, 정점} pair를 저장하는 min-heap
    C[1] = 1;                                       // 시작점 S = 1은 MST에 포함
    for(auto [i,w] : G[1]) pq.emplace(w, i);        // S에서 뻗어 나가는 간선들 Heap에 삽입
    while(!pq.empty()){
        auto [c,v] = pq.top(); pq.pop();
        if(C[v]) continue;                          // heap에 같은 정점이 여러 번 들어갈 수 있으니 주의
        C[v] = 1; ret += c;                         // v를 MST에 삽입
        for(auto [i,w] : G[v]) pq.emplace(w, i);    // v에서 뻗어 나가는 간선 정보 반영
    }
    return ret;
}
```

Time Complexity

- for(auto [i,w] : G[v])의 총 반복 횟수는 O(E) -> Heap에 원소 O(E)번 삽입
  - 각 정점마다 한 번씩 들어가므로, sum(deg(i))번 반복함
- Heap의 크기는 최대 O(E)이므로 시간 복잡도는 O(E log E)

- 참고
  - Heap에 각 정점마다 원소가 최대 한 개씩 존재하도록 구현하면 O(E log V)
  - Heap의 decrease key 연산을 O(1)에 구현하면 O(E + V log V)도 가능 (Fibonacci Heap)

Prim's Algorithm with Thin Heap

- O(E + V log V)를 사용하고 싶다면 GCC Extensions를 사용하자

```cpp
#include <bits/extc++.h>
using namespace std;
using PII = pair<int, int>;
constexpr int INF = 0x3f3f3f3f;

int N, M, C[10101], D[10101];
vector<PII> G[10101];

int Prim(){
    int ret = 0;
    __gnu_pbds::priority_queue<PII, greater<>, __gnu_pbds::thin_heap_tag> pq;
    vector<decltype(pq)::point_iterator> iter(N+1);
    for(int i=2; i<=N; i++) iter[i] = pq.push(PII(D[i] = INF, i));
    C[1] = 1;
    for(auto [i,w] : G[1]) pq.modify(iter[i], PII(D[i] = w, i));
    while(!pq.empty()){
        auto [c,v] = pq.top(); pq.pop();
        if(C[v]) continue;
        C[v] = 1; ret += c;
        for(auto [i,w] : G[v]) if(!C[i] && D[i] > w) pq.modify(iter[i], PII(D[i] = w, i));
    }
    return ret;
}
```

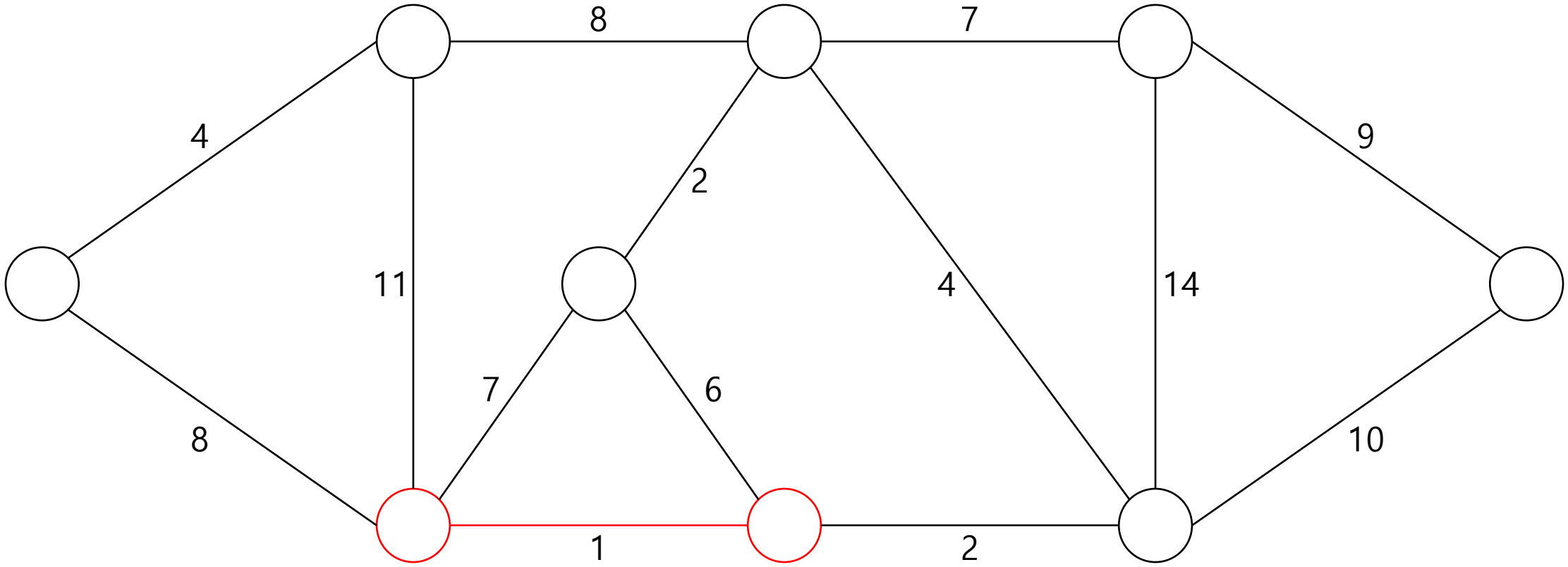
질문?

Kruskal's Algorithm

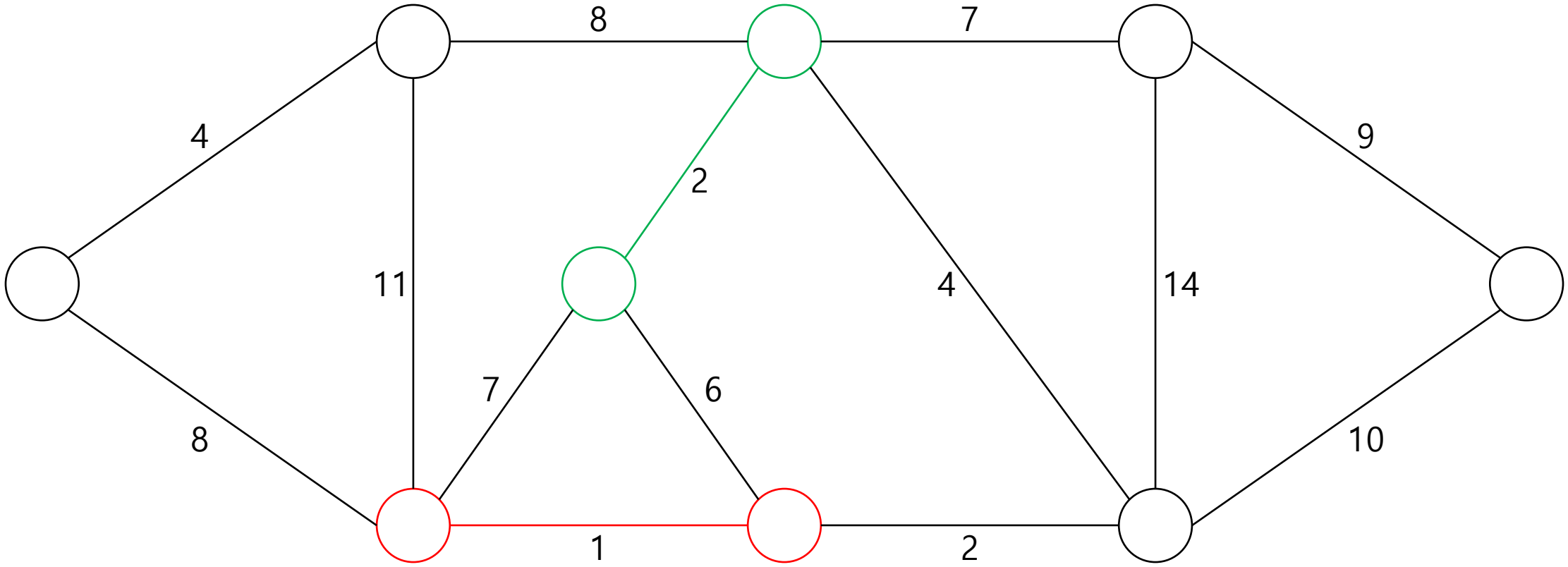
Kruskal's Algorithm

- 시간 복잡도: O(E log E)
- Spanning Tree에 간선을 하나씩 포함시키는 방식
- 그리디 기반 알고리즘
  1. 모든 간선을 가중치 오름차순으로 정렬
  2. 가중치가 작은 간선부터 보면서, 사이클이 생기지 않는다면 해당 간선을 MST에 추가
     - Prim's Algorithm과 다르게 알고리즘 진행 도중 트리가 여러 개 있을 수 있음
     - e = (u, v)에서 u와 v가 같은 트리에 있는지 확인해야 함 -> Union-Find

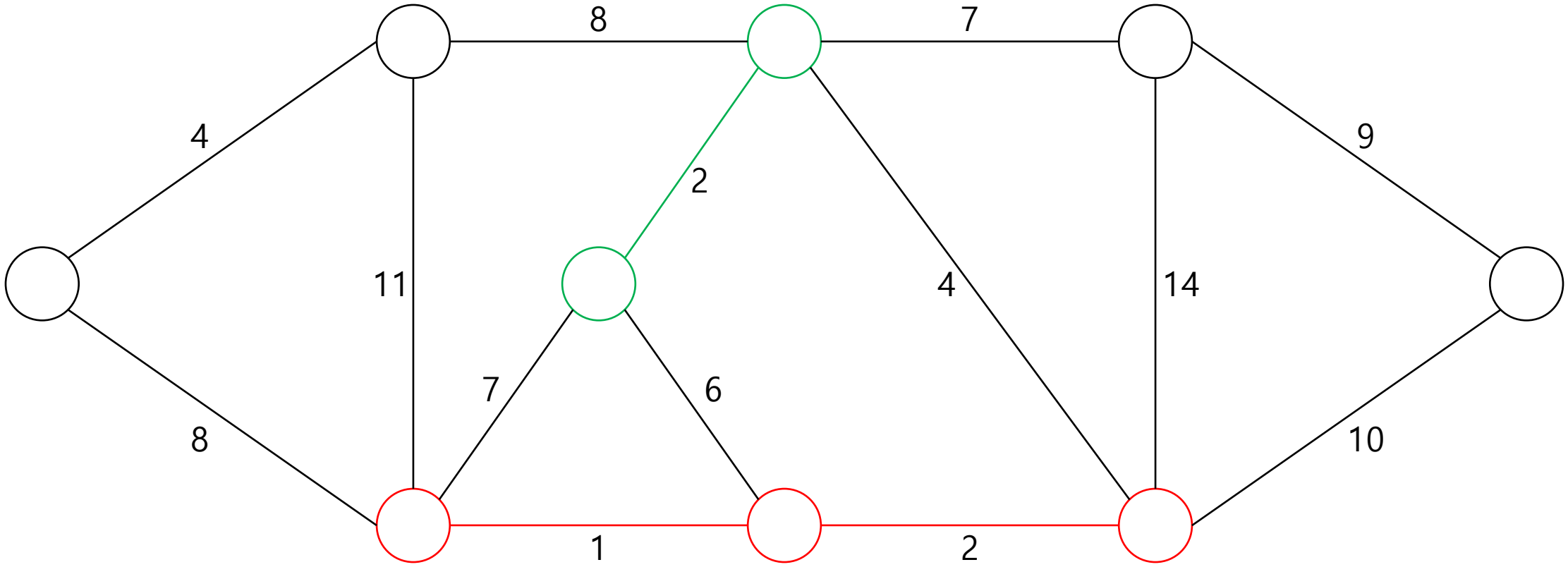
Example (1)



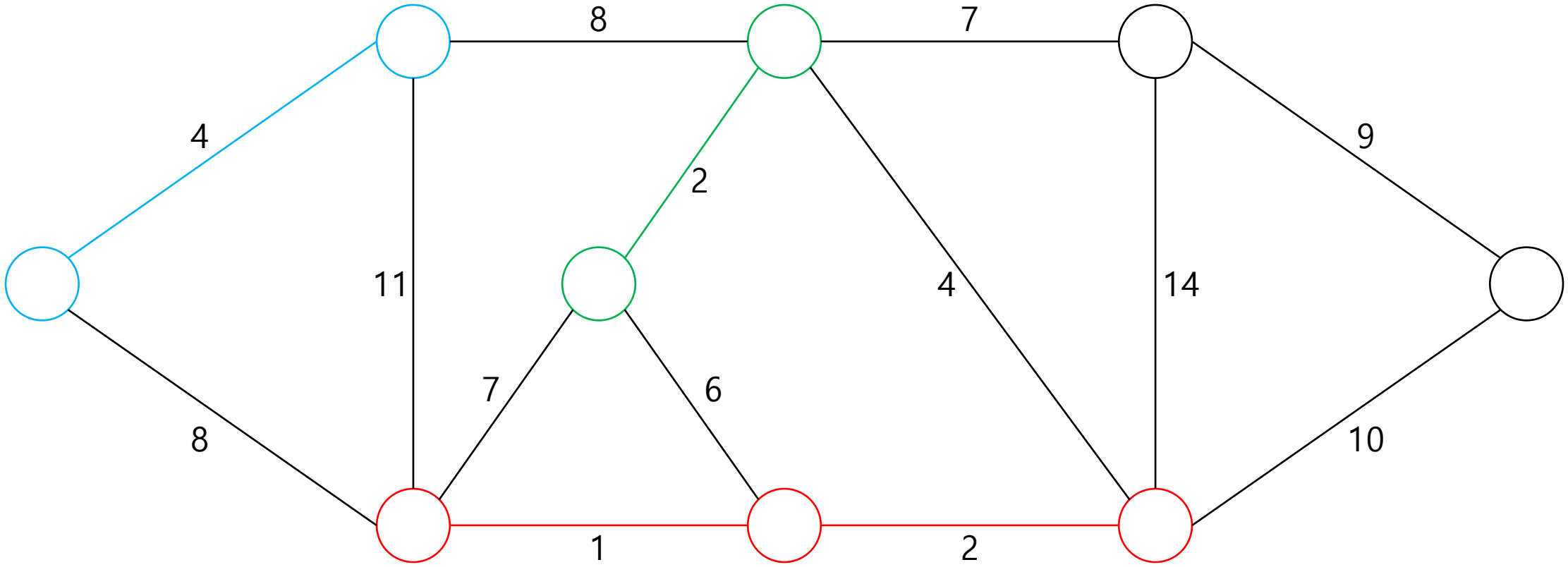
Example (2)



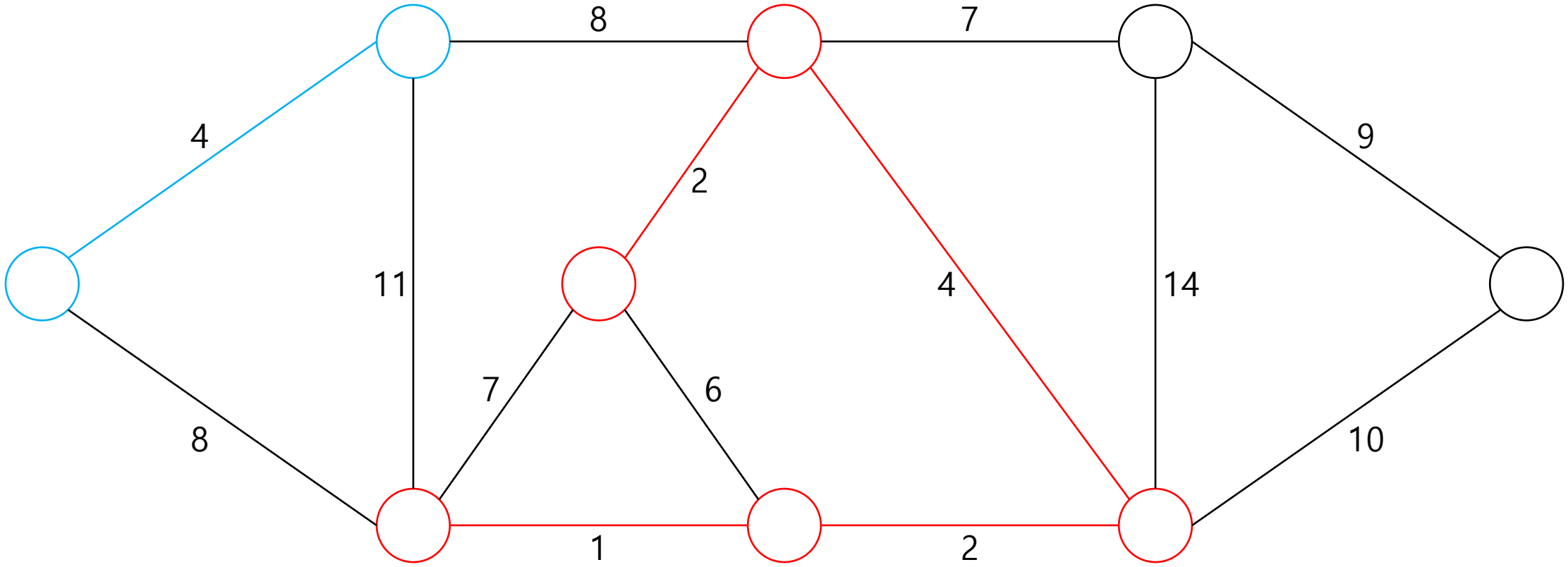
Example (3)



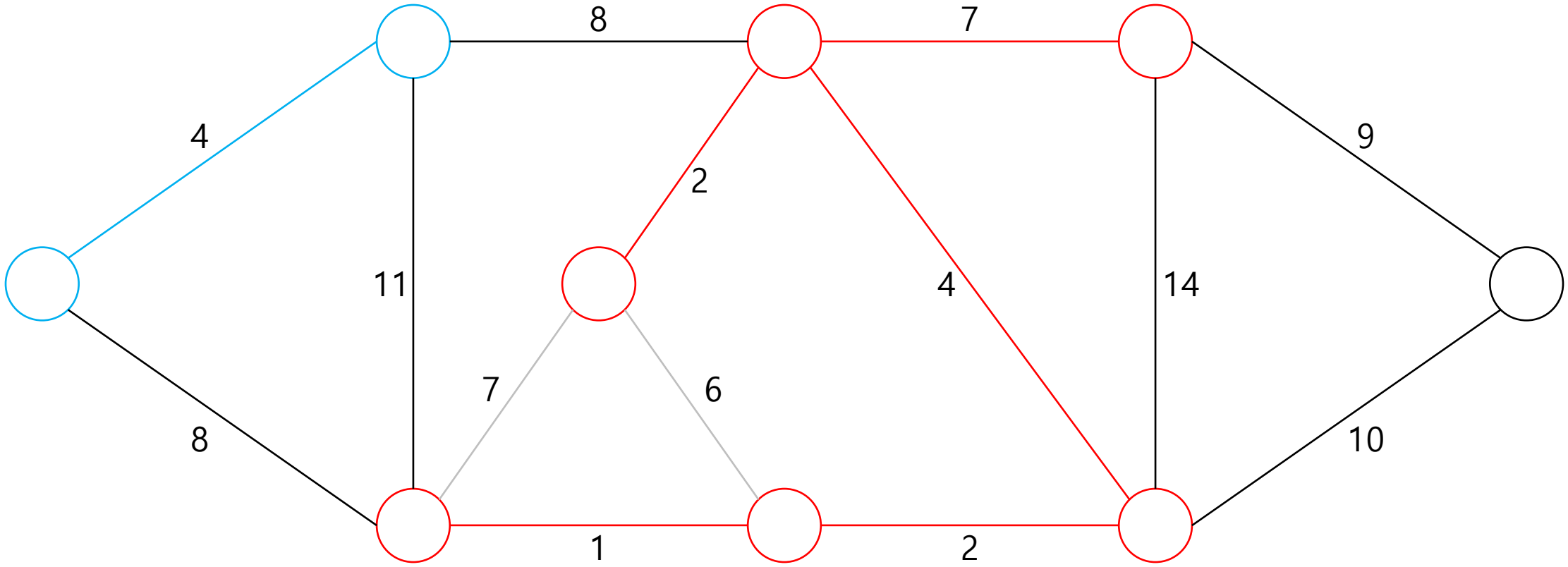
Example (4)



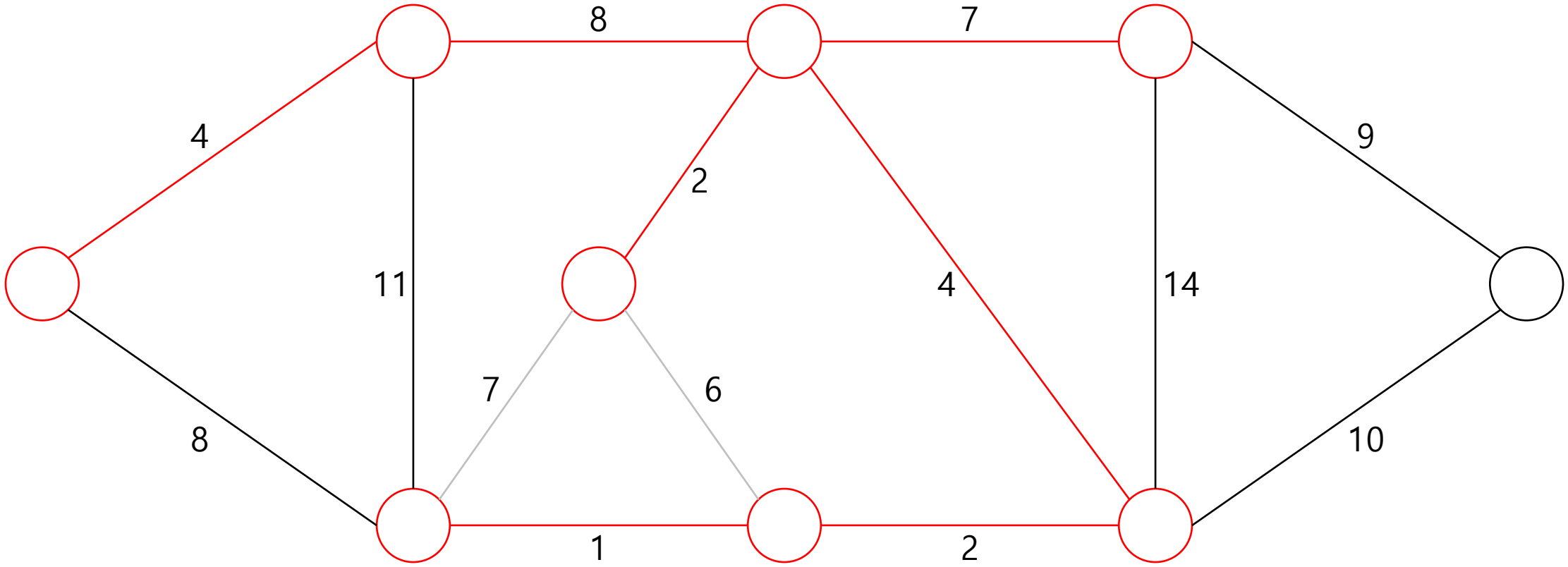
Example (5)



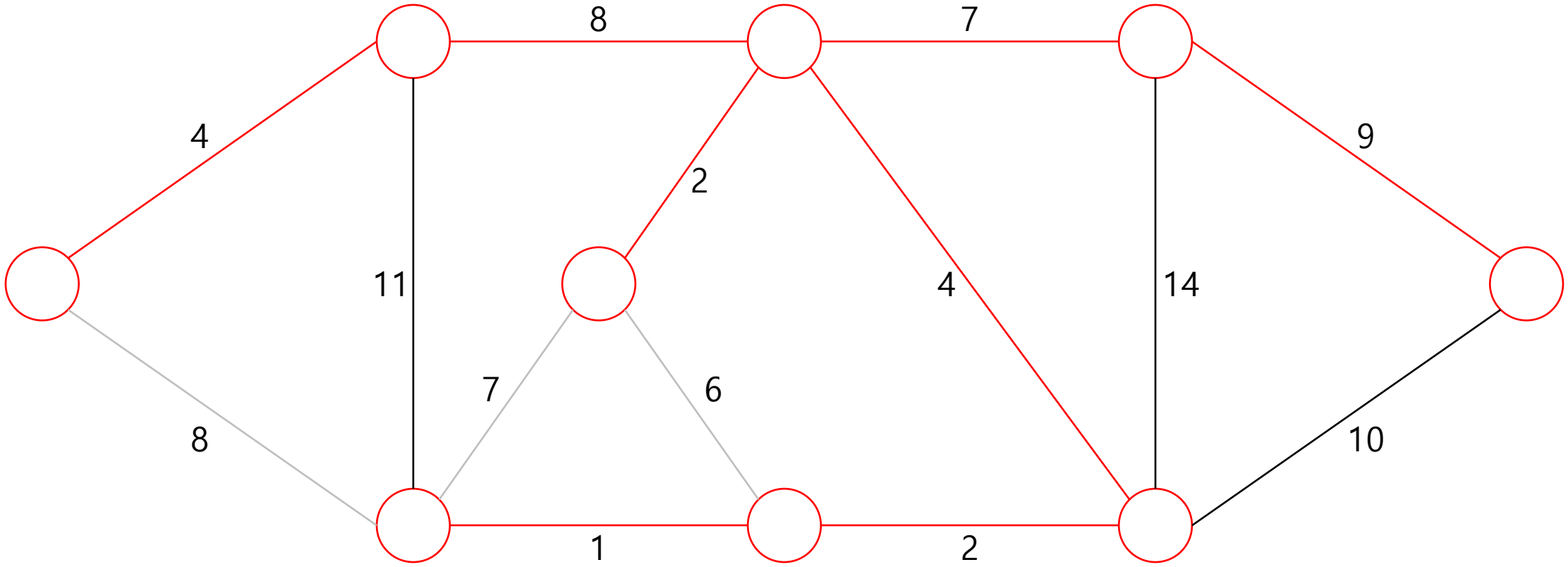
Example (6)



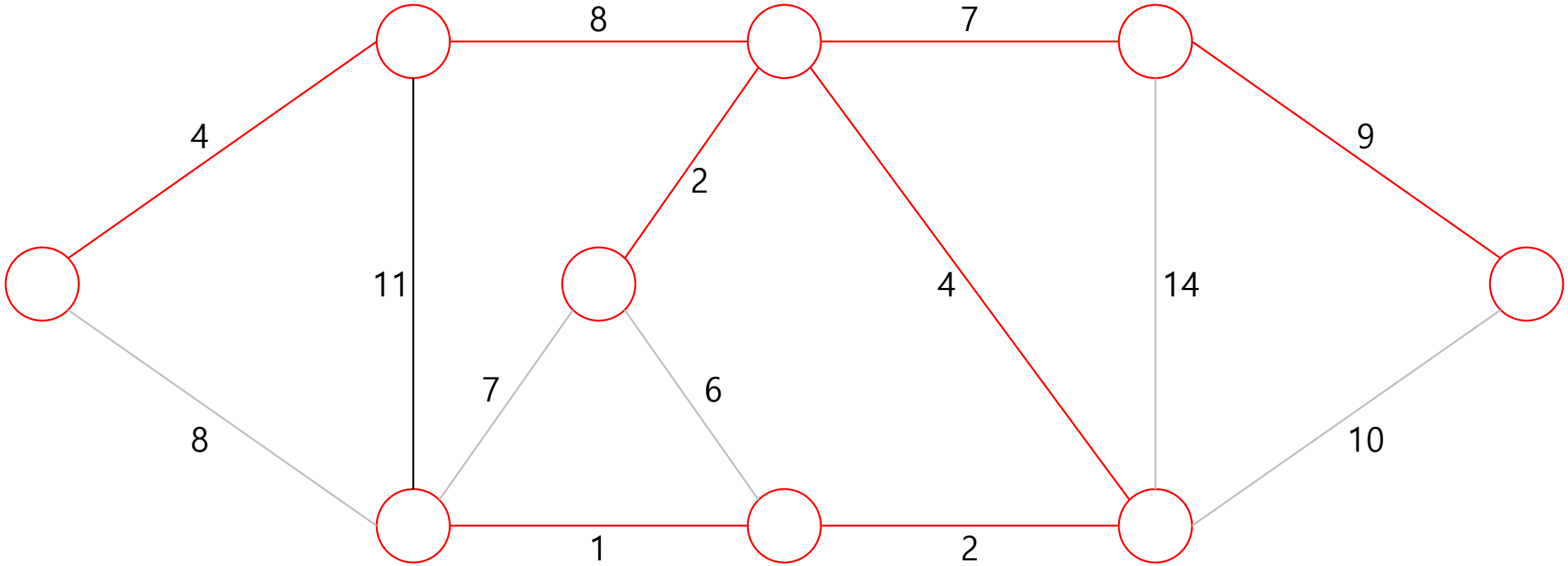
Example (7)



Example (8)



Example (9)



Kruskal's Algorithm

```cpp
struct Edge{
    int s, e, c;
    Edge() = default;
    Edge(int s, int e, int c) : s(s), e(e), c(c) {}
    bool operator < (const Edge &e) const {
        return c < e.c;
    }
};

int N, M, P[10101];
vector<Edge> E;

int Find(int v){ return v == P[v] ? v : P[v] = Find(P[v]); }
bool Union(int u, int v){
    u = Find(u); v = Find(v);
    if(u == v) return false;
    P[u] = v; return true;
}

int Kruskal(){
    int ret = 0;
    for(int i=1; i<=N; i++) P[i] = i;                // Union-Find 초기화
    sort(E.begin(), E.end());                        // 간선 오름차순으로 정렬
    for(auto [s,e,c] : E) if(Union(s, e)) ret += c; // 만약 사이클이 생기지 않는다면 MST에 삽입
    return ret;
}
```

Proof of Correctness

- 1학기 수업 5차시 슬라이드 마지막 10페이지 참고

Time Complexity

- 간선 정렬 : O(E log E)
- Union-Find 연산 : O(log V)짜리 연산을 O(E)번 수행
- 시간 복잡도 : O(E log E)

질문?

Prim vs Kruskal

- 구현: Kruskal이 쉬움
- 속도: Kruskal이 빠름
- Prim: 왜 씀?

- 간선의 가중치가 정점 번호에 대한 수식으로 표현되고, 메모리 제한이 작은 경우
  - O(V^2)짜리 Prim's Algorithm은 공간 복잡도 O(V)
  - Kruskal's Algorithm은 공간 복잡도 O(E)
  - BOJ 20390 완전그래프의 최소 스패닝 트리

Applications

- 최대 신장 트리
  - 가중치에 -1 곱하고 Kruskal's Algorithm

- u에서 v로 가는 경로 상의 가중치 최댓값을 최소화
  - MST에서 경로 최댓값 쿼리
  - LCA(Sparse Table)이나 HLD 같은 걸로 처리하면 됨

고인물 이야기 한 스푼

- 방향 그래프에서도 MST를 정의할 수 있다(Directed MST).
  - 어떤 정점 S를 루트로 하는 최소 비용 Rooted Tree를 만드는 알고리즘
  - O(VE)은 할 만하고(solved.ac 다이아3), O((V+E) log E)는 매우 어려워 보임(루비)

- 간선이 추가/제거되는 쿼리가 주어질 때 MST를 관리(Dynamic MST)하는 문제
  - 간선이 추가되는 쿼리만 주어질 때는 Link/Cut Tree를 이용해 O(Q log N)에 온라인으로 가능 (Link/Cut Tree만 알면 누구나 할 수 있음)
  - 추가/삭제 쿼리가 모두 주어지면 분할 정복을 이용해 O(Q log^2 N)에 오프라인으로 가능 (Parallel Binary Search 비슷한 느낌으로…)

- 사실 Maximum Spanning Tree는 Weighted Graphic Matroid에서…(생략)

질문?